NAVAL POSTGRADUATE SCHOOL

Monterey, California

THESIS

A TIME SLOT ASSIGNMENT ALGORITHM
FOR A TDMA PACKET RADIO NETWORK

by

William Karl Tritchler

March 1983

Thesis Advisor: J. M. Wozencraft

SECURITY CLASSIFICATION OF THIS PAGE (*When Data Entered*)

REPORT DOCUMENTATION PAGE		READ INSTRUCTIONS BEFORE COMPLETING FORM
1. REPORT NUMBER	2. GOVT ACCESSION NO.	3. RECIPIENT'S CATALOG NUMBER
4. TITLE (*and Subtitle*) A Time Slot Assignment Algorithm for a TDMA Packet Radio Network		5. TYPE OF REPORT & PERIOD COVERED Master's Thesis; March 1983
		6. PERFORMING ORG. REPORT NUMBER
7. AUTHOR(s) William Karl Tritchler		8. CONTRACT OR GRANT NUMBER(s)
9. PERFORMING ORGANIZATION NAME AND ADDRESS Naval Postgraduate School Monterey, California 93940		10. PROGRAM ELEMENT, PROJECT, TASK AREA & WORK UNIT NUMBERS
11. CONTROLLING OFFICE NAME AND ADDRESS Naval Postgraduate School Monterey, California 93940		12. REPORT DATE March 1983
		13. NUMBER OF PAGES 155
14. MONITORING AGENCY NAME & ADDRESS(*if different from Controlling Office*)		15. SECURITY CLASS. (*of this report*) UNCLASSIFIED
		15a. DECLASSIFICATION/DOWNGRADING SCHEDULE

16. DISTRIBUTION STATEMENT (*of this Report*)

Approved for public release; distribution unlimited

17. DISTRIBUTION STATEMENT (*of the abstract entered in Block 20, if different from Report*)

18. SUPPLEMENTARY NOTES

The research reported herein was funded in part by the Naval Ocean Systems Center, San Diego, California 92278.

19. KEY WORDS (*Continue on reverse side if necessary and identify by block number*)

Packet Switching; Packet Radios; TDMA; CDMA; Communications Networks; Integrated Voice and Data Communications; Spread Spectrum Signalling; Dijkstra Algorithm

20. ABSTRACT (*Continue on reverse side if necessary and identify by block number*)

An algorithm for the assignment of time slots within a Time Division Multiple Access (TDMA) scheme for an integrated voice and packet radio network is implemented in, and studied by, a computer simulation. The slot assignment scheme is applied both to a static network, where "best path" routes are held constant, and also to a network where the "best path" routes are permitted to change dynamically during the simulation as communications capability at various nodes approaches saturation.

The Dijkstra algorithm is used to determine and modify "shortest distance" routes, and the sensitivity of performance to various parameters used in defining the link "distance function" is investigated. The major conclusion is that it is possible to route in a way that reduces the average energy transmitted per message without substantially decreasing the network throughput.

A Time Slot Assignment Algorithm for a TDMA Packet Radio Network

by

William Karl Tritchler
Captain, U. S. Marine Corps
B.S., University of Wisconsin, 1975

Submitted in partial fulfillment of the requirements for the degree of

MASTER OF SCIENCE IN ELECTRICAL ENGINEERING

from the

NAVAL POSTGRADUATE SCHOOL
March 1983

Thesis
T8123
c.1

ABSTRACT

An algorithm for the assignment of time slots within a Time Division Multiple Access (TDMA) scheme for an integrated voice and data packet radio network is implemented in, and studied by, a computer simulation. The slot assignment scheme is applied both to a static network, where "best path" routes are held constant, and also to a network where the "best path" routes are permitted to change dynamically during the simulation as communications capability at various nodes approaches saturation.

The Dijkstra algorithm is used to determine and modify "shortest distance" routes, and the sensitivity of performance to various parameters used in defining the link "distance function" is investigated. The major conclusion is that it is possible to route in a way that reduces the average energy transmitted per message without substantially decreasing the network throughput.

TABLE OF CONTENTS

LIST OF ABBREVIATIONS

NCS	- Net Control Station
RDF	- Radio Direction Finding
DCT	- Digital Communications Terminal
I/O	- Input/Output
LOS	- Line-of-Sight
NPS	- Naval Postgraduate School
MAB	- Marine Amphibious Brigade
TDMA	- Time Division Multiple Access
FDMA	- Frequency Division Multiple Access
VHF	- Very High Frequency
UHF	- Ultrahigh Frequency
SHF	- Superhigh Frequency
STAR	- Simulation of Tactical Alternative Responses
AJ	- Antijamming
LPI	- Low Probability of Intercept
CDMA	- Code Division Multiple Access
PN	- Pseudonoise
THSS	- Time Hopping Spread Spectrum
FHSS	- Frequency Hopping Spread Spectrum
DSSS	- Direct Sequence Spread Spectrum
FSR	- Feedback Shift Register
SAW	- Surface Acoustic Wave
MF	- Matched Filter
CCD	- Charge-Coupled Device
NSA	- National Security Agency
CMS	- Classified Material System
DOD	- Department of Defense
EOM	- End of Message
TASI	- Time Assignment Speech Interpolation
b	- bandwidth of the compressed information signal
W	- bandwidth of the transmitted spread spectrum signal
PG	- Processing Gain
L	- chips per bit

MAF	- Marine Amphibious Force
DARPA	- Defense Advanced Research Projects Agency
IRFS	- Initial Request for Service
RRFS	- Response Request for Service
FAN	- Final Assignment Notice

ACKNOWLEDGMENT

It has been a pleasure to work in close association with Professor John M. Wozencraft. His patient guidance and unselfish availability for discussion and concept development is greatly appreciated. His broad knowledge of communications in general, and the stimulating insights he offered concerning packet-switched communications in particular, served to integrate much of my studies here at NPS and made this entire effort a valuable and rewarding experience.

This work is dedicated to my wife, Chris, whose constant support made it all possible.

I. INTRODUCTION

A. GENERAL

The purpose of military communications is to provide the military commander with the ability to exercise command and control over his forces. Military communications systems must be reliable, responsive to user requirements, and should offer a measure of security to the information carried. The commander's communications requirements were satisfied for centuries through the use of couriers and various visual and acoustic means of communications. All of these communications techniques have a place in the overall military communications picture today. However during the last several decades there has been tremendous technological development which has driven a rapid evolution of tactics as new weapons and support systems have been fielded. Most tactical military communications today are carried by electrical or electronic devices, e.g. radio or telephone.

A basic radio communications system includes at least two parties and a channel of communications between them. The channel is a frequency, a band of frequencies, or perhaps a wire or optical fiber with a bandwidth large enough to accommodate the modulated signals exchanged by the parties. A communications circuit is established when one party

(the originator) effects communications with another party (the addressee) over a channel.

Tactical radio communications today are primarily hierarchical in nature. That is, the flow of information is usually up and down the chain of command from senior to subordinate and vice versa. Lateral links between adjacent units are usually limited and are not well defined in current military communications doctrine. Lateral links, when employed, are usually operated with multichannel radio equipment and serve to increase the total communications system flexibility by providing alternate communications paths.

Current military voice radio and record or data communications circuits are operated in one of the three modes described below.

1. Broadcast Operation

In the broadcast method of operation one station transmits and the other station(s) receive. The flow of information is in one direction only, however different stations may broadcast at different times.

2. Point-to-Point Operation

A point-to-point circuit is one in which two stations communicate directly with each other. Both stations may transmit and receive signals.

3. Net Operation

Two or more stations that use a common channel to communicate comprise a net. Note that a point-to-point circuit is technically a net, although a net usually has several members. Typically one station on the net is designated as the Net Control Station (NCS) and is responsible for controlling net operations and for maintaining net discipline to ensure orderly and efficient operations. In a "directed net" any station other than the NCS which has traffic to pass must first request permission from the NCS before it may transmit its message. The radio (or teletype) operator at the NCS thereby manually controls the flow of traffic within the net. Since all stations on the net share the same channel it may be possible for two or more net members to communicate directly with each other (i.e. point-to-point) if the NCS so approves. It is also possible for the NCS to authorize the net to operate as a "free net". In a free net any station may send traffic to any other station in the net whenever the channel is available. This method of operation may permit greater message throughput if the net has few stations or the messages are brief and the traffic load is light. As the traffic load increases and more stations join the net, the directed net mode of operation may be required to reduce confusion and to promote the orderly exchange of information.

Today the net control function is done by a radio operator, and the tactical message traffic is passed by an operator using

APC 125 voice radiotelephone procedure or by a radio teletype operator. The voice radio messages may either be actual conversations between commanders (or staff officers) or may be properly drafted and released written messages that are then transmitted by trained radio operators. In any event, transmitting a message via voice utilizes the channel for a much longer length of time than would be required to transmit the same message if it were reduced to a teletype message. It is desirable to limit the amount of time any station is transmitting for two very important reasons. First, the chance of being detected and located by enemy radio detection finding (RDF) equipment increases with the amount of time a station is transmitting. Second, since only one station may use the channel at a time it makes sense to keep transmissions as brief as possible to provide more time for the other stations to use the channel.

This does not imply that all voice message traffic can or should be reduced to teletype or digital data messages. Indeed there appears to be a clear and present requirement for commanders on the battlefield to be able at times to converse directly with seniors and subordinates via voice radio. Moreover it is not yet practical to provide every radio with a means of automated message entry, although the Marine Corps has made some progress in this direction with the recent development of the AN/PSC-2 Digital Communications Terminal (DCT). The DCT is a hand-held, programmable I/O and display

device. It will enable users rapidly to compose, edit, and display free text, pre-formatted messages, and graphics such as maps.

The development and integration of computers and microprocessors with communications terminal equipment can permit the net control functions to be automated.

The use of computers on the modern battlefield is not limited to communications equipment. As weapons and military equipment in general become more complicated and capable, computers will find increased application. Computers can be used to process and manage large quantities of information and can provide the commander and his staff accurate and timely information.

Military communications doctrine is constantly evolving as communications requirements change to support new tactics, equipment, and organizational structures. There is an ever increasing trend toward the development of digital communications equipment because digital communications networks offer great potential for providing rapid, reliable, and secure circuits of very high quality. These are precisely the types of circuits required for computer and data communications. Digital communications equipment easily accommodates the digital representation of information generated and used by computers. Thus it is no accident that the development of communications equipment in general is trending along this line.

B. PACKET RADIO

There has been considerable research conducted since the late 1960's concerning packet-switching. Packet radio technology is advancing rapidly and its eventual application to military communications appears to be inevitable. Packet radio utilizes packet-switched communications and typically operates on a multiple access radio channel to create a digital radio network. A packet radio network has the capability to provide greater message throughput than the tactical military communications presently in use, and is particularly well suited to carry computer communications and other digital information such as digitized voice or facsimile traffic.

Packet-switching was originally developed as a cost effective method of supporting computer communications. The traffic generated by computers is "bursty" in nature and has a low duty cycle. That is, computers generate traffic at very high rates, but the individual messages are relatively brief and infrequent, so that the messages may be visualized across time as short bursts of data separated by long periods of inactivity. Since the channel may be idle nearly all of the time, it would be a very inefficient utilization of resources to provide a separate dedicated channel between each pair of computers that may have occasional requirements to exchange data. It is reasonable instead to arrange several computers (or data terminals) in a communications network and

to devise a controlling protocol which allows all of these data terminals to share a common broadcast channel. It is also reasonable to create a unit of transmission, called a "packet", of some appropriate number of data bits and to let a packet or series of packets be used to represent a data message.

In a packet-switched network each packet may be of a fixed (variable in some implementations) length up to a maximum of perhaps a few thousand bits. Each packet contains all of the addressing and control information necessary to route the packet to the desired destination. The addressing and control information might not be necessary in the follow-on packets of the packet-switching scheme employing virtual circuits. This will be discussed in later sections of this thesis.

The ability to connect any two network subscribers is an essential attribute of any communications network. If the packet radio equipment is designed in such a way that each packet radio may act as a relay or repeater in addition to the obvious requirement of being able to provide message entry and reception for local users, then it is not necessary for each terminal to communicate directly with every other terminal in the network. In the extreme, most of the packet radio terminals may be "hidden" from each other either because of the lack of a line-of-sight (LOS) path caused by intervening terrain and/or vegetation or because of radio range limitations. If

we assume that each packet radio has a very short range as compared to the diameter of the network, then all that is necessary for the connectivity requirement to be satisfied is that there exist at least one path, via any number of intermediate repeaters, between any pair of packet radios. A small computer or microprocessor is resident within each packet radio to implement a given packet-switching protocol or message routing scheme in a manner that is completely transparent to the user. This gives the user in the network the illusion of being directly connected to every other user in the network.

This is the basic idea of a packet-switched packet radio network. The network is composed of several compatible computer or microprocessor controlled radios operating on the same frequency or band of frequencies. Each radio communicates directly with one or more other network members, and has the capability both to service local users and to act as a repeater as required to provide full connectivity throughout the network as a whole.

Many packet-switching routing algorithms and multiple access techniques have been developed. The particular routing scheme and multiple access technique for use on a particular packet radio network should only be selected after a careful analysis of such questions as the type of broadcast channel, channel bandwidth, number of stations in the network, expected number of messages, message length, network topology and

connectivity, radiated power, signal energy, radio interference, microprocessor capability, propagation and processing time delays, the permissible message delay and so forth. Packet routing and multiple access techniques will be discussed further in later sections of this report. A brief summary of some of the previous research conducted at the Naval Postgraduate School (NPS) concerning packet radio is provided in paragraph C below.

C. SUMMARY OF PAST RESEARCH IN PACKET-SWITCHING CONDUCTED AT NPS

Considerable research has been performed recently at NPS on various aspects of packet-switching, and eight Master's Degree theses have been produced on the subject during the last three years. The author obtained much of his background information concerning packet-switching from these documents. A brief synopsis of each of these reports is provided in the following paragraphs.

Lucke [Ref. 1] studied the nature of distributed communications systems and their possible application to military communications. He discussed schemes for the distributed control of communications networks, routing strategies, and conducted a computer simulation of an asynchronous routing algorithm originally proposed by Segall and Merlin [Ref. 2]. He also devised a procedure for the time synchronization of a packet radio network.

Bond [Ref. 3] investigated the problem of self-interference in a packet radio network. He modeled the voice radio and record communications traffic load of a Marine Amphibious Brigade (MAB) and used this data in a computer simulation of a packet radio network to study the problem of self-interference. His routing algorithm dispatched messages over the path that required the fewest number of transmissions. Bond concluded that the MAB network must operate with either a Time Division Multiple Access (TDMA) or Frequency Division Multiple Access (FDMA) scheme in order to limit self-interference.

Kane [Ref. 4] studied the possible use of the VHF, UHF, and SHF frequency bands for tactical military packet radio communications. His work included the simulated tactical placement of a MAB on the STAR Terrain Model, a computerized parametric terrain representation of the Fulda Gap region in West Germany. The Simulation of Tactical Alternative Responses (STAR) Terrain Model was developed by Professor J. Hartman at NPS and is resident in the NPS IBM 3033 computer. Kane concluded that a packet radio network could be operated on terrain typical of western Europe. He proposed the employment of packet radios capable of operating at center frequencies of about 300 MHz for foliage penetration and at 1.5 GHz for increased channel capacity and decreased probability of interception.

Hobbs [Ref. 5] also used the MAB and STAR Terrain Model first studied by Bond and Kane to model the effect of superimposing a UHF "backbone" sub-network on the overall VHF MAB mobile distributed communications network. He developed two algorithms for creating connectivity topologies for the backbone and mobile sub-networks and concluded that it was possible to design robustly interconnected communications networks for the use of packet radio technology in the field.

Chlebik [Ref. 6] used the MAB topology and link connectivity developed by Hobbs to study by computer simulation the problem of mutual interference in a packet radio network. His simulations implemented the Dijkstra and Warshall-Floyd algorithms to determine minimum-hop paths between nodes, and included a study of the effect of using directional as well as omnidirectional antennas. He found that although mutual interference in the backbone sub-network was substantial, it was manageable. However, more than half of the lower frequency mobile nodes experienced unacceptably high levels of mutual interference much of the time.

Mercer's research [Ref. 7] entailed further study of routing schemes and their effects on interference in a packet radio network. He employed the MAB and STAR terrain models investigated earlier by Bond, Kane, and Chlebik and concentrated his efforts on comparing network performance with respect to the interference characteristics of least-hop and least-energy routing schemes. He concluded that least-energy

routing, or that perhaps a hybrid routing algorithm based on least-energy scheme, offered the best solution to the mutual interference problem.

Lengerich [Ref. 8] used computer simulations to evaluate the relative performance of two distributed routing protocols. In his work Lengerich specifically studied the Dijkstra shortest path routing algorithm and both a synchronous and asynchronous implementation of the Heritsch [Ref. 9: pp. 46-90] distributed dynamic routing scheme.

Heritsch [Ref. 9] devised and investigated by computer simulation a distributed routing protocol for a packet network. To reduce the size of the routing problem in large nets, he organized the nodes into Basic Groups, Related Groups, and Families, and created a network management protocol which demonstrated that efficient decentralized control of a packet radio network was possible.

D. PURPOSE AND SCOPE OF RESEARCH

1. General

Time division multiple access (TDMA) techniques and principles and their application to communications networks are well understood. There are many ways to implement a TDMA network. The different TDMA schemes offer varying degrees of efficiency, preservation of network flexibility, and conservation of overall network channel capacity. The particular type of TDMA scheme selected for implementation in any given

network will depend on many of the same considerations listed earlier for selection of a packet-switching scheme. However, TDMA scheme selection must also be based on the ability of the network to maintain time synchronization. The synchronization problem is addressed in section II.

2. Purpose

The purpose of this thesis is to develop and study by computer simulation a TDMA time slot assignment scheme appropriate for application to a packet radio network utilizing dynamic routing. The Dijkstra shortest path algorithm is used to periodically determine and modify "best path" routes between every pair of radios in the network. The performance of any network is highly dependent upon the type of "distance function" that is used to calculate the "link weights" which the Dijkstra algorithm uses to update the "best path" traffic routing tables. Accordingly, the research goals include a study of the sensitivity of performance with respect to various parameters used in calculating the distance function.

3. Scope

It was not possible or practical to simulate all of the time slot assignment schemes that were developed during the preliminary stages of research for this thesis. Time constraints and the amount of work required to write a simulation program demanded that we study only one or, at most, two slot assignment algorithms. We decided to

concentrate our efforts on two schemes that intuitively seemed to offer the greatest possible performance in a hypothetical military packet radio network. Both schemes were simulated on a small, richly connected packet radio network with static best path routing. One of the slot assignment algorithms gave performance substantially better than the other algorithm. Since it was reasonable to assume that the better algorithm would also yield superior performance when the simulation program was modified to accommodate dynamic routing, the poorer performing scheme was discarded and will not be discussed further. The remainder of this thesis is based on the research conducted with the better algorithm. This narrowed the scope of the thesis to one possible TDMA slot assignment scheme which could be thoroughly investigated in the available time.

It was necessary throughout the course of our studies occasionally to make assumptions concerning the design and operation of the hypothetical network which was being modeled. All of these assumptions (discussed in section III) somewhat limited the scope of the thesis. Assumptions, when required, were made after careful consideration of state of the art capabilities. The hypothetical network design and operating characteristics were developed based on what we believe are reasonable assumptions and opinions of how a military tactical packet radio network might someday operate.

II. PACKET RADIO NETWORK CONCEPTS

A. TERMINOLOGY AND DEFINITIONS

Packet radio has a vocabulary all its own. Some of the terms come from the branch of mathematics known as Graph Theory while the other terms are unique to communications or have no specific source. Before proceeding further it is necessary to provide the reader with definitions or explanations of some of the more frequently used terms found in the packet-switching literature and later portions of this report. Defined below are some of the terms essential for the discussion of basic network concepts. Other terms will be defined as required.

A "packet" is a unit of digital data of some fixed or variable number of bits. The packet radio network discussed herein utilizes fixed 192 bit packets; however it is possible to operate a network with variable length packets. Each packet usually contains a "header" which holds all of the routing and control information necessary to route the packet to its intended destination. A message is usually composed of many packets. The outgoing message is processed within the local (originating) packet radio or switch to divide the message into packets. The packets are then sequentially transmitted over the communications channel.

"Packet-switching" is the communications technique which connotes that there is individual packet processing at each packet radio or switch in the network in such a way that the packet's route through the network may be determined dynamically. In packet-switching each packet is transmitted from node to node across the network from the originator to the destination. As mentioned earlier, each packet switch may provide service to one or more local subscribers in addition to relaying through traffic.

A packet radio or switch is commonly called a "node", and the communications path between any pair of adjacent nodes is called a "link". The links in a network may be radio paths, wire trunks, or perhaps some combination of both of these. The network then is composed of nodes and links. As a brief aside, note that links may be unidirectional or bidirectional. Unidirectional links may be viewed as one-way streets or directed line segments while bidirectional links are analogous to two-way streets. Only bidirectional links were permitted in our hypothetical network because the time slot assignment algorithm required simplex communications between each pair of linked nodes in order to coordinate the assignment of time slots.

Each node in the network maintains one or more links with other network nodes called "neighbors". It is desirable for each node to claim more than one neighbor. This enhances

network connectivity, flexibility, capacity and overall reliability. It is not clear how many neighbors each node should try to claim or how many neighbors are sufficient to guarantee a measure of network robustness; it depends on such variables as the traffic load, equipment and path reliability, link capacity, terrain and radiated power constraints, whether the packet routing is dynamic or static, etc. There must be some practical bound on the number of neighbors a node would need or be able to claim. This is particularly true of our packet radio network implementation which required the assignment of a finite amount of equipment resources within each packet radio for each link to a neighbor. Hobbs work [Ref. 5] indicates that five or six neighbors per node produces attractive networks in typical situations.

A "weight" may be thought of as a cost. "Distance" and "channel value" are synonyms for weight frequently encountered in the literature. In our network we assign a "link weight" to each link. The link weight is a function of the link attenuation and therefore the energy per bit required to establish communications over the link. The low attenuation links are more desirable and are assigned a correspondingly lower weight than the less desirable higher attenuation links. We also assign a "node weight" which is a function of congestion present at the nodes on a link. The node weight

increases as one or both of the nodes on a link become more congested. The calculation of node and link weights is discussed in detail in section IV.

When a packet is transmitted over a link it is said to have made one "hop". A packet may traverse a single hop or multiple-hop path from an originator to an intended addressee depending on network connectivity and the proximity of the two communicating nodes. The link weight is used by the routing algorithm to determine what is referred to as the "best path" between any pair of nodes in the network. We seek to direct packet messages over the path that presents the least total cost. Link and node weight functions may be constructed that cause link and node weights to be calculated in such a way that the best paths are actually the least-hop or least-energy paths. It is also possible to design the weighting functions and perform the distance calculations to permit best path assignments based on a combination of least-hop and least-energy path considerations.

Time division multiple access (TDMA) is a signalling method by which two or more separate and distinct information bearing signals are transmitted over the same channel by allocating different time intervals for the transmission of each signal. TDMA permits all nodes in the network to share a common channel by transmitting signals that are separated in time. Our network used TDMA. Time was divided into time "frames". The frames had a fixed time duration. Each frame

was then divided into a number of uniform fixed length time "slots". Each slot could then be assigned to carry one packet.

Frequency division multiple access (FDMA) is a signalling method by which two or more separate and distinct information bearing signals may be simultaneously transmitted over the same communications path by sending each signal over a different carrier frequency. The possible implementation of a military packet radio network using FDMA was considered during the early stages of our research but was discarded because an FDMA network appeared to require a larger number of more complex receiver-transmitters than an equivalent TDMA implementation. Additionally, we decided early-on to use a spread spectrum technique to provide the packet radio transmissions the antijamming (AJ) and low probability of intercept (LPI) that spread spectrum communications offer. Although it seemed possible to devise a frequency hopping spread spectrum FDMA scheme, before such a scheme could be effectively implemented we would have to solve the same time synchronization problem which was the only major drawback to a direct sequence spread spectrum TDMA implementation. The time synchronization problem is addressed in paragraph D below, and once this problem was solved TDMA became the operating method of choice.

Code division multiple access (CDMA) is a digital communications technique that permits several separate and distinct signals to be transmitted and unambiguously received

over one broad-band channel at the same time. Each node in the network has assigned to it a unique pseudonoise (PN) code that may be thought of as specifying the node's address. The PN code is modulated by the outgoing binary data. CDMA is used in our proposed packet radio network because of the "selective addressing" capability it offers and because the CDMA technique is easily implemented in a spread spectrum communications network.

B. SPREAD SPECTRUM COMMUNICATIONS

Spread Spectrum is a communications technique that involves expanding the bandwidth of the information bearing signal. The expanded (spread spectrum) signal is then transmitted over a much wider range of the frequency spectrum than a more conventional signal with a transmitted bandwidth approximately equal to the bandwidth of the information. The desired signal is recovered by remapping the received spread spectrum signal into the original information bandwidth.

In a spread spectrum communications system the bandwidth of the data signal may be increased by one of three possible methods known as time hopping spread spectrum (THSS), frequency hopping spread spectrum (FHSS), or by a technique known as direct sequence spread spectrum (DSSS). It is also possible to design a hybrid spread spectrum communications system that employs two of these methods simultaneously. All of this is discussed fully in Reference 10, and since our

hypothetical network utilizes DSSS, the THSS and FHSS methods will not be discussed further.

The CDMA technique is readily implemented in a DSSS communications system. The "DS" in DSSS stands for "direct sequence", which refers to the high rate (large bandwidth) binary code sequence that is modulated by the lower rate data stream to produce a very wideband signal suitable for spread spectrum communications. It is possible to find PN code sequences with a low enough crosscorrelation so that CDMA communications are possible and the mutual interference is acceptable. One class of PN sequences can be easily generated by a programmable or a permanently wired feedback shift register (FSR). The modulated wideband signal is obtained by modulo two addition of the PN code and the outgoing data signal [Ref. 10: p. 5]. All of the packets transmitted by a node, whether locally generated or relay traffic, are modulo two added to the node's PN code sequence to produce the wideband signal that is then transmitted.

The received wideband signal must be processed at the receiving node to recover the baseband data which is then either delivered to a local subscriber or, in the case of relay traffic, used to modulate this node's PN sequence to produce a new wideband signal that is retransmitted on the link to the next node along the best path to the addressee. The received signal is applied to a bank of some type of correlation devices which reduce the signal to its baseband

form. The correlators may be surface acoustic wave (SAW) devices, programmable matched filters (MF), or programmable charge-coupled devices (CCD). In any event, each node must have one correlator set up and dedicated for use in receiving signals from each of its neighbors.

In addition to the multiple access and selective addressing capabilities already discussed, spread spectrum communications offer other advantages that are valuable in a military communications system. Earlier we alluded to the antijamming (AJ) and low probability of intercept (LPI) properties of spread spectrum systems. The wideband signal spectra produced in a DSSS system preferrably has its signal power spread uniformly across a wide band of frequencies. Therefore, the transmitted signal power density over any small range of frequency can be made quite small, perhaps 10 dB to 30 dB below the level of the background noise. Thus a spread spectrum signal may be buried in the background noise where it is not detectable with a conventional receiver.

Today our military codes and cryptographic devices and their associated keying material are controlled and distributed from the National Security Agency (NSA) through the Classified Material System (CMS) of the Department of Defense (DOD). If a number of PN codes with suitable crosscorrelation properties could be generated and then distributed through the CMS, and if the codes were changed frequently and properly protected by the local holders of the codes, then a military

DSSS packet radio network might not require additional cryptographic protection because the modulated PN bit stream exhibits the pseudorandom characteristic produced by any "good" cryptographic system. If additional cryptographic protection were required, then the data could be encrypted before being used to modulate the PN sequence to produce a cryptographically secure wideband signal for transmission. In this case a relaying node would receive and correlate the incoming wideband packet to collapse this incoming signal to an encrypted baseband signal. The encrypted baseband signal would then have to be processed by a cryptographic device connected to (or resident within) the packet radio to produce the plain-text baseband information packet. The node could then read the packet header and, seeing that the packet is destined for some other node, the relaying node would re-encrypt the packet and use the resulting data stream to modulate its own PN sequence to produce the spread spectrum signal it would then transmit to its best path neighbor on the link to the intended destination.

It might be desirable to leave the packet header unencrypted. Then a node would obtain a plain-text header with address information directly from the correlator. The node would then decrypt the remainder of the packets that were addressed to it, or would re-modulate and retransmit the packets addressed to other nodes without first decrypting and then re-encrypting.

In this thesis we do not study if or how a military packet radio network would be made secure by cryptographic devices. The methods proposed above are involved and admittedly equipment intensive; however, even if devices such as these are not today physically realizable, in the author's opinion it should be possible to build this type of cryptographic equipment by the time a military packet radio network is ready to be fielded.

C. VIRTUAL CIRCUITS

Person-to-person digital voice communications require the nearly continuous use of a low-bandwidth channel, whereas the more bursty computer-to-computer traffic generally needs intermittent use of a high-bandwidth channel. A packet radio switch can reserve and release channel capacity as needed to satisfy these communications requirements. Our network was designed to accommodate both voice and data communications; however, the method by which each of these is handled is different.

Interactive voice communications must be processed on a real-time basis to be useful, whereas data communications are largely one-way and may be reassembled and stored at the receiving terminal for later review. End-to-end delays of more than 0.1 seconds in voice traffic start to become noticeable and should be avoided, while delays in data communications are more tolerable as long as all of the data

packets are eventually received by the addressee and can be properly reassembled to recreate the original message. Data packets may be received in any order but voice packets must be received in the order in which they are transmitted and with relatively uniform delay to be useful. Note also that bit errors and lost packets are intolerable in data communications and therefore the use of error detection and correction codes is usually required. However the occasional occurrence of a bit error or lost packet may not seriously degrade the performance of packet-switched voice communications because the human ear will detect the error and the listener will interpolate and understand what is being said [Ref. 11]. Under these considerations it is reasonable to use "virtual circuits" to carry voice communications and to use the "store-and-forward" technique for the transmission of data packets.

In our packet radio network a virtual circuit is constructed for each voice communications requirement at the time that demand is placed on the network. Each virtual circuit consists of a pair (one for transmitting and one for receiving) of time slots on each link along the best path from the calling to the called party. The slots associated with each virtual circuit are then reserved or temporarily assigned for the duration of the conversation. Kuo [Ref. 12: p. 140] points out that the use of "a virtual circuit approach, in which routes are selected on a session-by-session basis

(depending on link utilization and topological connectivity criteria)" is one method to maintain packet sequencing.

Virtual circuits have another advantage in that once established, the succeeding packets do not require a complete packet header because the nodes along the best path have recorded their slot assignments in routing and slot assignment tables and therefore "know" that, in the case of a relaying node, packets incoming form the originator-side in a certain slot should automatically be retransmitted a few milliseconds later to the best path neighbor in a specific slot that was reserved when the virtual circuit was established. The relaying node does the same thing for the packets in the other half of the conversation, i.e. the voice packets from the called to the calling party. Additionally, if we use a separate buffer or queue to temporarily store the voice and data packets as they await retransmission at relay nodes, then the voice packet queue may be very small because, according to our algorithm, a voice packet would never have to wait for more than 1 frame plus 1 slot duration before being retransmitted. However, the data packet queue would normally be much larger in order to hold the many data packets that could accumulate at a node that is becoming congested.

The algorithm that was developed to simulate the construction of virtual circuits is presented in section III; however some comments concerning how requirements for voice

communications would be placed on the proposed packet radio network are in order here. It is envisioned that a virtual voice circuit in a packet radio network could be constructed in much the same way that typical telephone (circuit-switched) communications are established today. A caller would use a combination handset and keypad to "dial" the party with whom voice communications are desired. It seems likely that a tactical packet radio network should be able to interface with the tactical telephone system to provide trunking on an as required basis, and thus provide telephone subscribers with the capability to direct-dial any other telephone subscriber in the integrated wire and packet radio network. The speaker's voice would be digitized within the handset and then packetized within the local packet radio. In any event, after the calling party identifies the called party the packet radios automatically attempt to build the virtual circuit along the best path according to the routing and slot assignment protocols. The caller is then provided with and audio and/or visual "busy" or "ring" signal. The busy signal might indicate that the called party was already engaged in conversation with someone else or that a link along the best path could not accommodate the assignment of a pair of mutually available time slots. The calling party would then re-dial the call at some later time.

Once the virtual circuit is established, either party may signal the end of the circuit requirement, i.e. "hang up",

by pressing or releasing a key on the handset or by returning a telephone handset to its cradle. The packet radios then automatically break down the virtual circuit from the party who first hung up to the other party. The time required to build or break down a virtual circuit depends, in part, on the slot assignment protocol that is used; however, it seems reasonable to expect that even multiple-hop circuits can be built or rebuffed as busy in much less than 1 second. Established circuits can be broken down very easily because the slots are already assigned and available to carry an end of message (EOM) indicator.

In our network the voice communications circuits take precedence over data communications requirements because of the requirement that voice communication be real-time. Data packets are passed one link at a time as slots and channel capacity are available. The data packets are examined for errors as they are received at each node. The reception of a correct data packet may be acknowledged to the neighbor node that sent the packet. Similarly, a node may request retransmission of a data packet with detected errors. Once a correct packet is received, it is placed in a data queue to await transmission to the next node along the best path to the addressee. This is known as "store-and-forward" operation. So the data packets may be thought of as filler traffic that is transmitted between voice virtual circuits. The virtual circuits are always built when the required slots are available.

It is interesting to note that studies have shown [Ref. 13] that the average speaker in a two party conversation is only actively vocalizing approximately 40 percent of the time. Speakers talk in "talkspurts" of activity separated by pauses to breathe and listen. It may therefore be possible to send data packets between the talkspurts of a conversation. A technique such as this called Time Assignment Speech Interpolation (TASI) has been used since 1960 to nearly double the usefulness of expensive deep sea telephone cable systems. [Ref. 14]

D. NETWORK TIMING AND SYNCHRONIZATION

1. Overview

Our packet radio network is designed to operate synchronously. Synchronous operation here means that all nodes in the network use frames that are synchronous in time. The time duration of any frame (or slot) is the same everywhere in the network thus eliminating the need for more capable "gateway" nodes to link sub-networks employing different frame/slot structures and timing.

Our network is homogeneous. All nodes are equally capable. In military parlance it could be said that the packet radios are standardized, interoperable and easily interchangeable. These are all desirable qualities of a military communications system because they lead to enhanced system flexibility and reliability, and serve to reduce the

obvious vulnerability of a network which employs a few highly sophisticated nodes, the destruction of which would seriously degrade network performance.

Although it is possible to simulate our slot assignment algorithm in a network where, at any instant, two interconnected nodes may be at the start of different slots, the requirement that all the nodes be effectively synchronized with respect to time slots is, in our network, absolute. Our additional requirement that the frames be synchronized is not unrealistic. It certainly makes the computer simulation program easier to write and also allows network operation and program execution to be much more easily traced.

We now consider whether it is technically possible to synchronize the proposed network in such a way that all neighboring nodes in the network start the same numbered slot of a frame at very nearly the same time. The analysis below is a reasonable first approximation concerning the timing requirements of our network. The analysis is laced with key assumptions of how a military integrated voice and data packet radio network might operate.

2. Analysis

The first matter to be settled is the selection of an operating frequency. Kane [Ref. 4] suggested that a military packet radio network should be operated as two sub-networks with different operating frequencies. He proposed that most of the packet radios operate at 300 MHz to permit greater

network connectivity. Kane also recommended that a "backbone" sub-network operating at 1.5 GHz be superimposed on the 300 MHz network to provide greater bandwidth and correspondingly greater message carrying capability. This system would require some type of interface equipment between the two sub-networks. Additionally, the 1.5 GHz radios needed line-of-sight (LOS) paths nearly free of vegetation because of the highly directional and poor foliage penetration properties of signals at this frequency. On the battlefield this requirement means that the backbone terminals would usually be sited on high ground relatively free of cover where they could be vulnerable to enemy observation. Therefore, we decided that our network would operate as if its frequency were about 300 MHz where LOS paths were less critical and adequate connectivity had been demonstrated by Kane.

The analysis that follows is based on the assumption that our network utilizes delta modulation (DM). Readers unfamiliar with DM may wish to consult Reference 15, pp. 539-546 or Reference 16, pp. 498-506. DM is used extensively in military communications equipment being developed by the Joint Tactical Communications Office under the TRI-TAC program.

Delta modulated voice communications are typically sampled at 16 kilobits per second (16 kbps). If the voice circuits are operated as "virtual circuits", and if we allow only one slot per frame to be assigned for the transmission

or reception of a particular voice circuit, then for a system with 12 uniform slots per frame the duty cycle for any single circuit is

Duty Cycle = 1/12 = 0.0833

If we assume that each time slot has a duration of 1 millisecond then each twelve slot frame is 12 milliseconds long.

Each virtual circuit must pass traffic at an overall rate of 16 kbps, and since the duty cycle is 0.0833, this implies that the information in each voice virtual circuit must be compressed by a factor of twelve. Therefore, in our twelve slot per frame scheme, the information in each slot must be passed at a rate of,

(16 kpbs)/(0.0833) = (16 kbps)(12) = 192 kbps

Thus the bandwidth b of the compressed (lowpass) signal which will be used to modulate the PN code sequence is,

b = 192 kbps = 192 kHz

Since each slot is 1 millisecond long, each assigned slot much carry,

(192 kbps)(1 ms) = 192 bits

of the compressed information.

Assuming that our network operates spread spectrum at a center frequency of approximately 300 MHz, then as a rule of thumb we could reasonably expect to spread our signal over a radio frequency (RF) bandwidth W equal to about one half of one tenth of the operating center frequency. Thus

$$W = (300\ \text{MHz})/(2)(10) = 15\ \text{MHz}$$

Therefore the PN sequence rate is 15 Mbps and the post detection processing gain (PG) of the spread spectrum signal is approximately

$$PG = W/b = (15\ \text{MHz})/(192\ \text{kHz}) = 78.125 = 18.9\ \text{dB}$$

In spread spectrum terminology the bits in the high rate PN sequence are called "chips", and as discussed earlier, the chip sequence is modulated by the data to produce the spread signal. The chip rate is the same as the bandwidth W of the spread signal. The number of chips L per modulating data bit is also given by

$$L = W/b = (15\ \text{MHz})/(192\ \text{kHz}) = 78.125\ \text{chips/bit}$$

In any actual spread spectrum implementation we would require a whole number of chips per bit. Therefore we would round off the result of this calculation to 78 chips per data bit, which would change slightly the bandwidths and PG calculated above. In other words, we would select some integral number of chips per bit that would yield the desired spread spectrum bandwidth and PG.

In a military packet radio network we would want to use the most compact and inexpensive oscillator that would satisfy our timing requirements. In our proposed packet radio network we seek to synchronize the timing signal derived from the local oscillator to within 0.1 chip of the received chip stream in order to properly correlate the received signal. That is, the incoming chip stream must be synchronized in time with the locally generated reference PN sequence that is used to remove the effects of spreading (i.e. correlate) and reduce the received signal to its compressed baseband equivalent.

Oscillator performance is measured in terms of an oscillator's short-term and long-term stability characteristics. Short-term stability refers to the oscillator's ability to "beat" regularly over a brief period (perhaps 1 second) of time while long-term stability is a measure of oscillator accuracy measured over a much longer period (usually hours or days).

Today the oscillators or frequency standards available commercially fall into two general categories: quartz devices and atomic frequency standards. The frequency produced by a quartz oscillator is the result of vibrations originating in the piezoelectric nature of the quartz crystal itself. The frequency of an atomic standard is derived from the energy transition between atomic states that is an

intrinsic characteristic of the atom involved. Some typical values for oscillator and frequency standard stability are given in References 17 and 18.

Atomic frequency standards have very good long-term stability and slightly poorer short-term stability. In contrast, quartz oscillators have very good short-term stability but drift in frequency over the long-term. In further contrast, the operating frequencies of quarts oscillators range in value from 0.1 to 100 MHz while the atomic standards resonate at much higher frequencies in the range of 6.8 to 9.2 GHz, and quartz oscillators are generally smaller, lighter, require less power to operate, and cost much less than atomic frequency standards [Ref. 17]. Therefore we would prefer to use crystal oscillators if at all possible.

We must now decide what degree of stability is required for the oscillators in our network. Once this is decided we will be in a position to determine whether or not our proposed network is physically and economically realizable.

If we let the oscillator frequency be ten times the chip rate then,

$$\text{Oscillator Frequency} = (10)(15\text{ MHz}) = 150\text{ MHz}$$

The careful reader will note that we said crystal oscillators have a frequency limit of 100 MHz. However, recent developments in crystal oscillator technology have extended the

operating range to 300 MHz for "standard type" and up to 1 GHz for "custom-designed type" oscillators [Ref. 18].

Then in one of our 12 ms frames there will be

$$\begin{aligned} \text{Oscillations per Frame} &= (150\ \text{MHz})(0.012\ \text{sec}) \\ &= 1.8 \times 10^{6} \end{aligned}$$

beats of our oscillator in each frame. If we were able to resynchronize our oscillator once each frame, then we would require an oscillator with a short-term stability of about

$$\text{Short-Term Stability} = 1/(1.8 \times 10^{6}) = 5.56 \times 10^{-7}$$

or, said another way, a stability of about six beats in every 1×10^{7} beats of the oscillator. Direct extension of this result leads to the development of the information applicable to our network presented in Table I.

Typical crystal oscillators have short-term stability of about 1.5×10^{-11} over 0.01 seconds and 1×10^{-11} over 100 seconds with long-term stabilities on the order of 5×10^{-10} over a twenty-four hour period [Ref. 17]. The new "standard type" quartz oscillators offer even better short-term stabilities of 1×10^{-10} to 1×10^{-12} over 1 second [Ref. 18].

It appears as though our network would require occasional global resynchronization with a master clock. We would prefer to perform this global resynchronization as infrequently as possible to keep the network management and

overhead traffic at a minimum. The method by which global resynchronization can best be accomplished has not been studied as a part of this thesis. However, it seems reasonable to perform this function during and in conjunction with the best path update cycles.

TABLE I Short-Term Stability Requirements

Resynchronization Period	Resynchronization Rate	Short-Term Stability Required
0.012 sec	Once per frame	5.56×10^{-7}
0.120 sec	Once per 10 frames	5.56×10^{-8}
1.200 sec	Once per 100 frames	5.56×10^{-9}
12.00 sec	Once per 1000 frames	5.56×10^{-10}
120.0 sec	Once per 10000 frames	5.56×10^{-11}

Utilizing a conservative crystal oscillator short-term stability estimate of 1×10^{-10} and interpolating from Table I we see that resynchronization is only required about once every six seconds. This is close to the update periods studied in later sections of this thesis. Thus is it reasonable to conclude that timing and synchronization should be achievable in our proposed network with crystal oscillators.

III. A PROPOSED TDMA TIME SLOT ASSIGNMENT ALGORITHM

A. ASSUMPTIONS

As our hypothetical network model was developed and refined, it was necessary to make several key assumptions concerning the way in which a military packet radio network might someday operate. The most important assumptions are discussed in the following paragraphs.

The modeled network is shown in Figure 1. In designing our test network we sought to devise a network that was simple to implement and large enough to generate the dynamic conditions that might be encountered in an actual packet radio network. The test network contains thirteen nodes and thirty links, and can be called "richly connected". The network connectivity was assumed to be static. No nodes were permitted to join or leave the network and all of the links were assumed to remain intact for the duration of the simulation. The nodes were assumed to be located approximately 3 kilometers to 6 kilometers apart. Therefore propatation delays would be on the order of 10 to 20 microseconds and were regarded as neligible.

The results of the analysis presented earlier in this report allowed us to assume that timing and synchronization

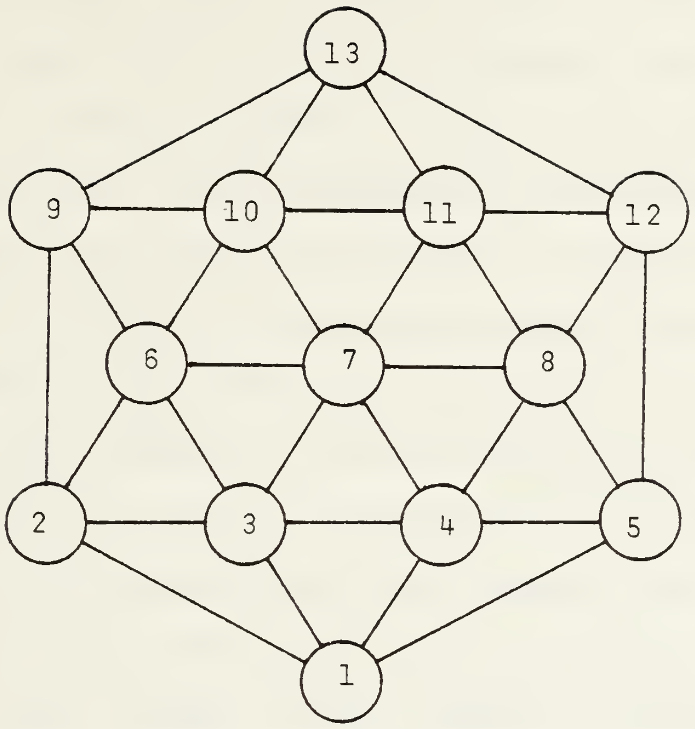


Figure 1 Test Network

could be achieved in our network. Additionally, we assumed that the noise and interference characteristics of the channel were such that intelligible voice communications could always be effected (subject, of course, to link, node, and slot availability).

Our network was presumed to be heterogeneous, that is, capable of accommodating both real-time voice messages and data traffic. Bond's work [Ref. 3: pp. 24-46] included an analysis of voice and data traffic requirements based on historical data of a Marine Amphibious Force (MAF) deployed

in Vietnam. He used this data and the information contained in a recent Marine Corps Tactical Systems Support Activity (MCTSSA) study [Ref. 19] to conclude that, in the future, voice radio communications within a MAB will be "the major contributor to network loading". Since we decided to use virtual circuits for voice messages and the store-and-forward technique for data packets, and since we assumed that the volume of voice traffic would greatly exceed the total amount of data traffic, it was decided to restrict our simulation to studying only the effects of using virtual circuits. Therefore, the simulated flow of data packets was omitted from our study. If we consider that data packets can be buffered in queues within the nodes and sent when channel capacity is available (either between virtual circuit requirements and/or in the interstices between talkspurts of an established virtual circuit), then it is reasonable to assume that our network could also easily process a relatively light load of data packets.

All links were assumed to be bidirectional and both halves of a conversation were carried by the same link or series of links.

Hobbs work [Ref. 5: pp. 20-24] included a study of the link equations for a prototype tactical packet radio network laid out in central Europe. His networks have characteristics and features that are similar to the network we developed. Hobbs concluded that for a typical network, with the packet

radios utilizing omnidirectional antennas, it is possible to establish a communications network with enough alternate routes to provide reliable operation using links whose loss (i.e. attenuation) does not exceed approximately 141 dB. Hobbs also found that the best path in his network layout had an attenuation of 81 dB. Thus it is reasonable to model our network with link losses that range in value from approximately 81 dB to 141 dB. We assigned a randomly selected attenuation in this range to each of the thirty links in our network. These link attenuations are contained in Appendix A.

We now postulate several basic operating rules for our packet radio network. First, a node may either transmit or receive in a slot but may not do both simultaneously, because when a node transmits, the transmitted signal effectively jams any signal that the node is attempting to receive. We also assumed that each node was "listening" for inter-nodal service messages in any slot in which it was not transmitting. Second, since CDMA operation was assumed, all nodes could receive packets from more than one neighbor simultaneously. How many packets could be received at once, i.e. the "depth" to which the nodes could "stack" the receive signals, was a program input parameter.

Finally, in order to make the simulation more manageable, we assumed all of the nodes in the network had instantaneous and global knowledge of all link and node weights whenever

a best path update was performed. This is an admitted artificiality, because in any actual packet radio network that utilizes dynamic routing there would have to be some type of update or network management protocol in operation to modify weights and generate and process update messages. It was beyond the scope of this thesis to devise or test an update scheme.

It is worth noting, however, that the ARPANET, a packet network designed and managed by the Defense Advanced Research Projects Agency (DARPA) of the DOD, utilizes a dynamic routing update protocol that has produced very good results [Ref. 20: pp. 226-231]. The topic of passing routing information in a distributed packet radio network is a subject of current research.

B. THE DIJKSTRA SHORTEST PATH ALGORITHM

Determination of the "shortest path" between any pair of nodes in a weighted graph is a classic problem that has been studied by mathematicians and graph theorists. As previously mentioned, this problem is directly applicable to communications networks.

Many algorithms have been developed to find the minimum weight path connecting a pair of specified nodes. The Bibliography of this report lists several textbooks that discuss the most popular shortest path algorithms.

The algorithms are usually very easy to implement on a computer. However, depending on the size, connectivity, and traffic flow constraints of the network, some of the algorithms may require very long computer execution times. Therefore several "heuristic" algorithms have also been developed. These algorithms generally provide sub-optimum solutions with much less computational effort.

Research by Gallager [Ref. 21] has proven that the paths in a minimum distance (optimum) solution, for a network with link weights greater than zero, is loop free. Although any optimum solution must be loop free, not every loop free solution is optimum. Therefore we sought an algorithm that was computationally easy and that would yield an optimum solution, thus providing efficient operation and loop free path assignments.

We selected an algorithm first described by Dijkstra [Ref. 22] for implementation on our network. The Dijkstra algorithm is basically a "tree growing" procedure wherein we substitute links, as required, into paths from every node to every other node on successive iterations of the algorithm. Each node must know the network topology and all of the link distances. The algorithm iterates until all of the minimum distance paths have been identified, that is, until we make a pass through the algorithm without making a change to the entries in the cumulative "distance" and "best path neighbor" routing tables.

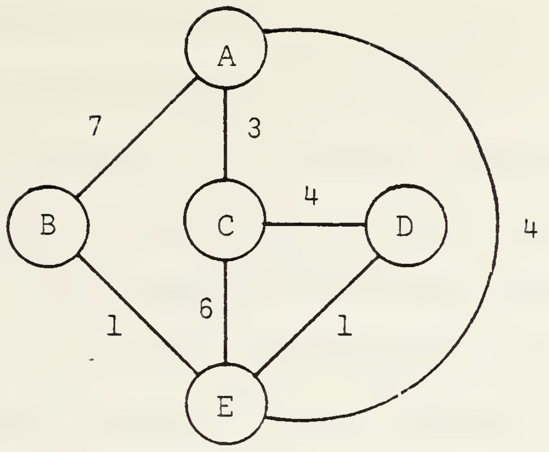
If we let d_{ij} represent the distance from node i to node j, then the actual operation of Dijkstra algorithm can be described as the successive calculation of

$$d_{ij} = \min[d_{ij} \text{ or } \min_k(d_{ik} + d_{kj})]$$

for each pair of nodes in the network.

The actual operation of the Dijkstra algorithm is best explained by example. Given the small network and initial distance and best path neighbor routing table in Figure 2, we shall demonstrate how the Dijkstra algorithm can be used to obtain the best path neighbor assignments. Nodes may be numbered or lettered. Here they are lettered to avoid confusion with the link distances.

The network of Figure 2 has five nodes lettered A through E and seven bidirectional links. The number beside each link represents the "distance" of that link. As discussed earlier, the link distance is not necessarily the physical distance between the nodes but rather is any positive number representing the cost of using that link. Although the algorithm may be used to find the minimum distance paths in networks with unidirectional, bidirectional, or a combination of unidirectional and bidirectional links having different link distances, we have for simplicity in our example assigned one distance for each link. That is to say, the distance between any pair of nodes on a direct link is the same in either direction.



Initial Best Path Neighbor

From \ To	A	B	C	D	E
A	-	B	C		E
B	A	-			E
C	A		-	D	E
D			C	-	E
E	A	B	C	D	-

Initial Distance

From \ To	A	B	C	D	E
A	-	7	3	∞	4
B	7	-	∞	∞	1
C	3	∞	-	4	6
D	∞	∞	4	-	1
E	4	1	6	1	-

Figure 2 Dijkstra Algorithm Example Network and Associated Tables

The initial best path neighbor and initial distance matrices contain only the neighbors and distances associated with the direct links. Note that the distance between nodes that are not directly connected is initially set to ∞, and that the best path neighbors for these node pairs are unassigned.

Beginning with node A, observe that a direct link exists with node B and that the distance from node A to node B is 7. We shall follow the notation A/B/7 (used by Lengerich [Ref. 8]) as a convenient means of describing the path and its distance. According to the algorithm, we next examine every other path from node A to node B to determine if a channel value less than 7 can be found. First consider the path A/C/3 + C/B/∞ which represents the two hop path A/C/B/∞ from node A to node C and then from node C to node B. Since the path from node C to node B has not yet been determined, the weight of this two hop path is infinite. The A/C/B/∞ path therefore is rejected and no changes are made to the best path or distance tables. Next the A/D/∞ + D/B/∞ = A/D/B/∞ path is considered and subsequently rejected because it also has an infinite distance. Finally the A/E/4 + E/B/1 = A/E/B/5 path is examined and adopted as the tentative new best path from node A to node B because the new resultant cumulative distance of 5 is less than the direct path distance of 7. The best path and distance tables must now be modified to reflect that node A's best path neighbor to node B is node E, and that the total path distance via node E is 5.

Next we look for cumulative distance paths from node A to node C which are lower than the direct link A/C/3. We consider the paths A/B/6 + B/C/∞ = A/B/C/∞, A/D/∞ + D/C/4 = A/D/C/∞, and A/E/4 + E/C/6 = A/E/C/10 and reject all of these paths.

Next we seek a tentative best path from node A to node D. There is no direct link between these nodes so the A/D/∞ path initially has an infinite distance. We consider the path A/B/6 + B/D/∞ = A/B/D/∞ and reject this path because if its infinite total distance. The A/C/3 + C/D/4 = A/C/D/7 is next studied and adopted as the new tentative best path with the tables modified accordingly. Therefore we are now looking for a path with a cumulative distance less than 7. The next path we consider, A/E/4 + E/D/1 = A/E/D/5 with a cumulative distance of 5 is just such a path and is therefore adopted as the new best path from node A to node D, and the table entries for A to D are modified to show that node E is the best path neighbor and that the distance of this path is 5.

The procedure outlined above is continued and at the end of the first pass through the tables the best path neighbors and distances are as shown in Figure 3.

Best Path Neighbor

From \ To	A	B	C	D	E
A	-	E	C	E	E
B	E	-	E	E	E
C	A	E	-	D	D
D	E	E	C	-	E
E	A	B	D	D	-

Distance

From \ To	A	B	C	D	E
A	-	5	3	5	4
B	5	-	7	2	1
C	3	7	-	4	5
D	5	2	4	-	1
E	4	1	5	1	-

Figure 3 Dijkstra Algorithm Example Table Values After the First Pass

Since changes were made to the tables during the first pass, we must now make a second pass through the tables to see if the changes made during the first path will permit still better path assignments. We make two changes during the second pass through the distance table, both associated with the path between nodes B and C.

During the second pass there are no changes until we seek a path from node B to node C with a cumulative distance less than the value of 7 (via node E) obtained on the first pass. The path B/D/2 + D/C/4 is really the path (B/E/1 + E/D/1) + D/C/4 = (B/E/D/2) + D/C/4 = B/E/D/C/6, which is a three hop path where the path shown in parentheses is a two hop path identified during the first pass. The distance of 6 corresponding to the newly identified three top path is less than the distance value 7 obtained earlier, so we modify the distance table accordingly. Note however that node B's best path neighbor assignment is still node E.

Later during the second pass we discover a similar change to the path from node C to node B. Considering the path C/D/4 + D/B/2, which is really the path C/D/4 + (D/E/1 + E/B/1) = C/D/4 + (D/E/B/2) = C/D/E/B/6 (where once again the path in parentheses is the two hop path identified during the first pass), we obtain a lower three hop path distance of value 7. Now, however, we must modify both the best path neighbor and distance matrices because C's best path neighbor has changes from node E to node D.

There are no more changes during the second pass, and after the second pass the best path neighbor and distance tables are as shown in Figure 4.

Best Path Neighbor

FROM \ To	A	B	C	D	E
A	-	E	C	E	E
B	E	-	E	E	E
C	A		-	D	D
D	E	E	C	-	E
E	A	B	D	D	-

Distance

From \ To	A	B	C	D	E
A	-	5	3	5	4
B	5	-	6	2	1
C	3	6	-	4	5
D	5	2	4	-	1
E	4	1	5	1	-

Figure 4 Dijkstra Algorithm Example Table Values After the Second Pass

Since there were changes during the second pass, a third pass through the tables is now required. We make no more changes during the third pass, so the algorithm terminates and we adopt as final the best path neighbor assignments contained in Figure 4.

Each node now need only know which neighboring node is its best path neighbor to every other node. In our network implementation we continue to route each existing virtual circuit to the best path neighbor (i.e. over the same path)

that existed when the virtual circuit was established. However any new virtual circuits and all data packet are now sent according to the updated best path neighbor assignments until such time as the Dijkstra algorithm is again invoked. Depending on the network topology, node mobility, and the function used to determine link distances, the shortest paths will change over time.

The routing information and assignments produced by the periodic execution of the Dijkstra algorithm as described above actually provides for "quasi-static", rather than truly "dynamic", routing because the best path neighbor assignments are held constant between updates. The link distances may change several times between updates and therefore truly dynamic routing would require that an update be performed each time a link distance is changed, which is clearly impractical in a network with more than a few nodes or in any network where the traffic volume and flow changes rapidly. If the period of time between updates is relatively short (perhaps on the order of 1 to 5 seconds) with respect to anticipated significant changes to the link distances, then it is reasonable to expect that quasi-static routing should perform nearly as well as truly dynamic routing. It is not uncommon to find quasi-static routing referred to as dynamic routing in the literature.

C. THE PROPOSED TIME SLOT ASSIGNMENT ALGORITHM

1. Design Goals

We are now ready to discuss the time slot assignment algorithm we have developed for a military packet radio network. Our design objectives are discussed briefly in the next several paragraphs.

We sought to devise a scheme that would use CDMA to allow two or more received signals to be "stacked" and simultaneously recieved in one time slot, thereby conserving empty slots (and channel capacity) and allowing greater throughput under conditions of heavy network loading.

Additionally, we sought a slot assignment scheme which would distribute the transmit signals across all slots of a frame as uniformly as possible over the network as a whole. This should maintain the overall radiated energy of the network at a relatively constant level over any frame (or short series of frames) and should also help to minimize the amount of mutual interference.

The use of a dedicated "service slot" to carry network management and virtual circuit coordination traffic was considered initially but later rejected as an inefficient allocation of channel capacity. We decided to design the algorithm in such a manner that inter-nodal communications coordination traffic is passed in any of the available slots.

A desirable algorithm should attempt to service all offered voice traffic, and the simulation program should take

into account and realistically model the delays encountered in an actual packet radio network. Although propagation delays were considered to be negligible, other time delays such as the time to process a packet and the time a processed packet must wait until being retransmitted were modeled in the simulation program.

Finally, we desired a slot selection algorithm that was easy to implement and compatible with the Dijkstra dynamic routing algorithm. This was not a problem. Our proposed time slot assignment algorithm should work well with any dynamic routing scheme and will detect looping and backtracking caused by changes to the best path neighbor assignments during the construction of virtual circuits.

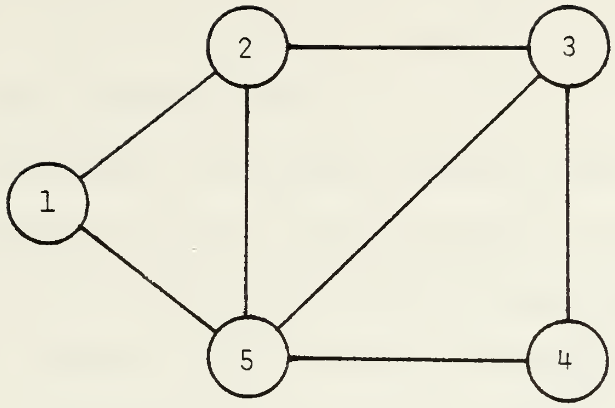
2. The Algorithm Explained

The basic premise of our time slot assignment algorithm is that the nodes should seek to conserve their unassigned slots by stacking the received signals whenever possible to some maximum depth in a minimum of slots. The stacking depth is a simulation program input parameter, however we require that all nodes always be able to receive one signal more than the assigned stacking depth. This requirement is necessitated by the fact that we have assumed that a node may always receive a communications coordination message from a neighbor in any slot in which it is not already transmitting.

Since every node is "listening" to its neighbors in any slot in which it is not already transmitting, a node may know a lot about its neighboring nodes' transmit slot assignments. It will not, however, know which slot or slots a neighbor is already using to receive. Therefore our algorithm uses brief single packet messages to coordinate assignment of the time slots. We let the node that is being called select and assign the slot in which it will receive a virtual circuit. A node makes a receive slot assignment based on the information in the calling node's coordination message and a knowledge of its present slot assignments. The requirement for neighbor nodes to exchange communications coordination messages is the reason for our earlier requirement that all links be bidirectional.

As with the Dijkstra algorithm already discussed, it is easiest to explain our time slot assignment algorithm with an example. We assume that our network is composed of five numbered nodes with the best path neighbor information and slot assignments as shown in Figure 5. Note that this example network uses four slots per frame rather than the twelve slots per frame scheme which was actually studied. However, four slots per frame is sufficient to demonstrate the operation of the algorithm.

We shall assume that the best path neighbor assignments will remain as shown in Figure 5 for the duration of the simulation and that we seek to stack the receive signals



Best Path Neighbor

From \ To	1	2	3	4	5
1	-	2	5	5	5
2	1	-	3	3	5
3	5	2	-	4	5
4	5	3	3	-	5
5	1	2	3	4	-

Time Slot Assignment Tables

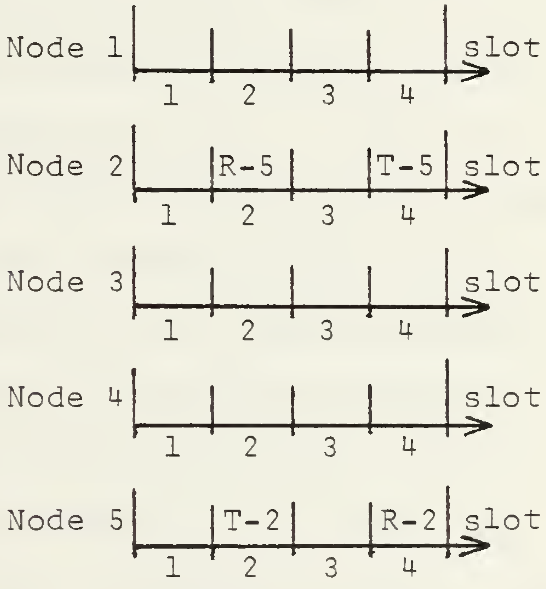


Figure 5 Time Slot Assignment Algorithm: Example Network and Associated Information

three deep (i.e. receive up to three signals simultaneously) when possible. We have adopted the notation T-3 and R-3 to signify that a slot is used to transmit to node 3 or to receive from node 3 respectively.

Before we start our example we see from the Time Slot Assignment Tables in Figure 5 that there is already a single hop virtual circuit established and actively carrying voice traffic between nodes 2 and 5. Node 2 is transmitting to node 5 in slot 4 and receiving from node 5 in slot 2. Similarly, node 5 is transmitting to node 2 in slot 2 and receiving from node 2 in slot 4.

We now begin the example by assuming that we are late in slot 1 when a caller at node 1 dials or somehow identifies a requirement to speak with someone at node 3. The packet radio that is node 1 recognizes the requirement for a virtual circuit and consults its best path neighbor table. Node 1 finds that its best path neighbor for all traffic destined for node 3 is node 5, and prepares an "initial request for service" (IRFS) message for transmission to node 5, but by now the whole network has just entered slot 2. Node 1 has been listening and knows that node 2 transmits in slot 4 and that node 5 (the node with which it must now communicate) is transmitting in slot 2. Since our rules prohibit a node from simultaneously transmitting and receiving, node 1 must wait and transmit the IRFS to node 5 in slot 3.

Node 5 receives node 1's single packet IRFS message in slot 3, consults its time slot assignment table, and sees that its slots would best be conserved if it could receive node 1's transmissions in slot 4 (i.e. in the slot already used to receive transmissions from node 2). Node 1's IRFS included information concerning its present slot assignments, so node 5 knows that node 1 is able to transmit in slot 4. Therefore node 5 assigns slot 4 as the slot in which node 1 will transmit and node 5 will receive. Node 5 now prepares a "response request for service" (RRFS) message for transmission back to node 1, but because of the time required to process the IRFS the network is in slot 4 and node 5 must wait until slot 1 of the next frame to send its RRFS back tc node 1.

Node 1 receives node 5's RRFS in slot 1 and sees that it has been directed by node 5 to transmit in slot 4. Node 1 will now record this slot assignment and then, with the help of the slot assignment information provided in the RRFS, select a slot in which it will receive from node 5. Since node 1 has no other receive slots assigned but knows from the RRFS that node 5 is already transmitting in slot 2, node 1 may select either slot 1 or 3 for use in receiving from node 5. We shall assume that node 1 selects slot 3 as the receive slot. Node 1 records this assignment in its time slot assignment table and prepared a "final assignment notice" (FAN) message for transmission to node 5. The time required

to process the RRFS, select a receive slot, and produce the FAN message means that the network is now in slot 2. Node 1 now identifies the next slot which it may use to send the FAN to node 5. As with the IRFS this is slot 3. Note that node 1 could use its assigned transmit slot (slot 4) to carry the FAN if there were no available slots occurring earlier.

Node 5 receives node 1's FAN in slot 3 and records that node 1 has directed it to transmit in slot 3. The time slot assignment tables for nodes 1 and 5 now appear as shown in Figure 6. The nodes have now constructed one hop of the virtual circuit. Node 5 now starts building the next hop of the circuit.

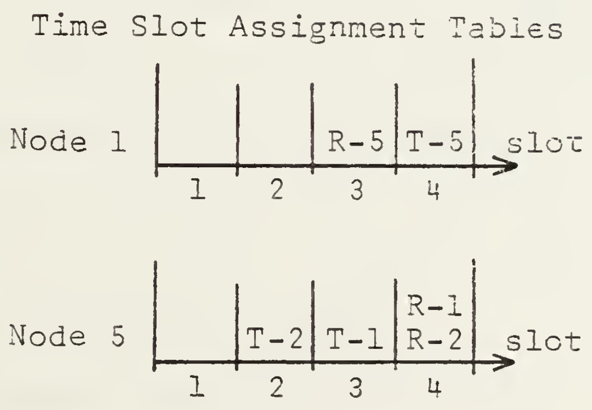


Figure 6 Time Slot Assignment Algorithm Example - Time Slot Assignments for Nodes 1 and 5

Node 5 knows that the virtual circuit addressee is node 3, so node 5 checks its best path neighbor table and sees that its best path neighbor to node 3 is node 3. The network is well into slot 4 by the time node 5 prepares an IRFS for transmission to node 3. Therefore node 5 waits until slot 1 (its nearest and only remaining unassigned slot) of the next frame to transmit its IRFS. Node 5 has been listening in slot 1 and knows that node 3 is not transmitting in this slot.

Node 3 receives and processes node 5's IRFS and determines that it must tell node 5 to transmit in slot 1 since this is node 5's only remaining free slot. Fortunately node 3's slot 1 is not already assigned as a transmit slot, nor is it receiving a maximum number of receive signals, or else our circuit requirement would have had to be rebuffed and the slot assignments associated with the first hop removed from the slot assignment tables at nodes 1 and 5.

Node 3 records that it will receive from node 5 in slot 1 and prepares an RRFS for transmission in the next mutually available slot, which node 3 identifies as slot 4. By now the network is in slot 2 so node 3 must wait until the start of slot 4 to send its RRFS.

Node 5 receives the RRFS from node 3 in slot 4, records that it will transmit to node 3 in slot 1, and after application of the time slot assignment algorithm decides

that it must receive from node 3 in slot 4. Node 5 now records this decision and also makes appropriate strap-over records, both for the purpose of effecting automatic retransmission of the traffic on this virtual circuit and also to facilitate the orderly and efficient breakdown of this circuit at a later date. Node 5 records that all traffic received from node 1 in slot 4 should automatically be retransmitted to node 3 in slot 1. Similarly, the traffic received from node 3 in slot 4 should be retransmitted to node 1 in slot 3.

The network is in slot 1 by the time node 5 completes all of the processing outlined above and drafts a FAN for transmission to node 3. Therefore node 5 must wait until slot 1 of the next frame to pass its FAN to node 3.

Node 3 receives node 5's FAN and records that it has been directed by node 5 to transmit in slot 4. The time slot assignment tables for nodes 1, 3, and 5 now appear as shown in Figure 7. Node 3 recognizes that it is the addressee for this circuit and sends a ring signal (or some other indication that an incoming call has been received) to a local subscriber or switchboard. Node 3 should also send a service message back to the originator over the circuit just established to let the calling party know that the virtual circuit has been constructed.

The virtual circuit between nodes 1 and 3 has now been established. Note that node 5 is saturated. It has no

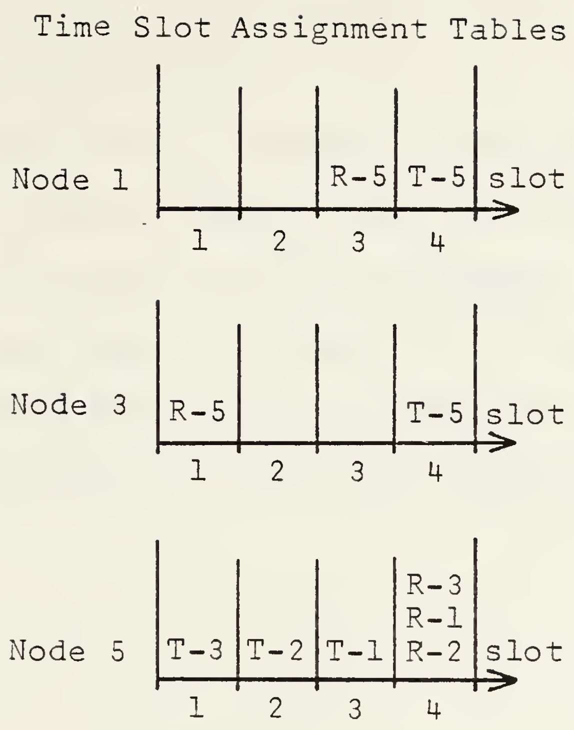


Figure 7 Time Slot Assignment Algorithm Example - Final Time Slot Assignments for Nodes 1, 3, and 5

unassigned slots and is receiving a maximum of three signals in its one receive slot. Assuming that there are no changes to the network between now and the next best path update cycle, the calculation of the distances for node 5's direct links should yield large values of distance, so that the new best paths are selected in such a way that future circuit requirements not originated at or addressed to node 5 are routed over the three links connecting nodes 1 and 2, 2 and 3, and

3 and 4. Node 5 should be avoided since all calls to node 5 will be rebuffed until one or both of the virtual circuits presently active at node 5 are disestablished.

It should now be clear to the reader that increasing the number of time slots per frame or increasing the maximum allowable receive signal stacking depth can have a significant impact on the overall message throughput. Equally obvious is the fact that, according to our rules, no node will ever be able to stack receive signals to a depth greater than the number of nodes it claims as neighbors.

IV. THE COMPUTER SIMULATION PROGRAM

A. COMPUTER LANGUAGE AND RESOURCES

The simulation program was written in the SIMSCRIPT II.5 programming language. The SIMSCRIPT language is versatile and has many features that make it well suited for discrete-event simulations. The language is relatively easy to use and SIMSCRIPT programs are (with a little practice) easy to read because the program statements are written in an approximation to simple English. The read and write statements may be "free-form" or formatted, and errors produce excellent diagnostic messages.

The simulation program was executed on the NPS IBM 3033 computer, running SIMSCRIPT II.5 version 9.0.

B. PACKET RADIO NETWORK SIMULATION PROGRAM

The simulation program has a modular design. In addition to the "preamble" and "main" program, there are nine "events" and eight "routines". SIMSCRIPT routines are basically the same as subroutines in other programming languages. Each routine performs a specific function and may be called by the main program, other routines, or any event anytime during the simulation. Events differ from routines in that events are "scheduled" rather than called. The main program and any event or routine may schedule any event to occur at the

present time or some future point in time. (References to time in this section of the report refer to the modeled simulation time maintained by the computer's simulation clock during program execution.)

Copies of the simulation program and a sample data set are appended to this thesis. The program contains ample comments and each event and routine carries a header of comments to help explain its purpose and function.

1. Distance Calculations

A distance (i.e. cost) function is used to calculate the link distances, which are then used by the routing algorithm to determine the best paths. The distance function may consider path attenuation, link and node congestion, packet delay time, queue length, etc. The distance function will normally consider and attempt to interrelate several of these parameters in order to produce distances which, when operated on by the dynamic routing protocol, produce desirable path assignments.

Kuo [Ref. 12: p. 163] states that: "There is no universally optimal routing strategy". If delay is important in a particular network, then the distance function should produce weights that assure route selections which avoid pockets of local congestion. If the amount of radiated energy is important, then the distance function should produce weights which will yield least-energy routing.

We think of adaptive routing as a congestion avoidance mechanism. However, Kuo [Ref. 12: p. 20] also points out that this is only true if the congestion is local. If the congestion is a symptom of excessive traffic entering the entire network, then dynamic routing just serves to spread the congestion. Networks use flow control procedures to regulate the amount of traffic entering the communications network. Flow control procedures are not discussed in this thesis.

The distance function in our simulation is composed of two principal computations, that is, each complete "link distance" is obtained by adding a "node weight" and a "link weight".

The link weight is solely a function of the link attenuation. As previously mentioned, each of the thirty links was assigned an attenuation between 81 dB and 141 dB. The program assigned each link attenuation to one of 128 "link weight bins". The links were assigned to the bins according to a geometric distribution. The lowest attenuation link was assigned to bin number 1, while the highest attenuation link was assigned to bin number 128. The remaining links were interspersed in the other bins. The attenuation bin assignments are contained in the appended Sample Input Data.

The link weight is obtained by identifying which bin the link is in. We use the link's bin number as its link

weight. For example, the link in bin number 60 has a link weight of 60. Thus, the link weights range in value from 1 to 128, with the majority of links assigned to the lower numbered bins because of the geometric link distribution.

The "node weight" is more difficult to obtain. It is primarily a function of how busy the nodes at each end of the link are. Each node compares the number of its slots in current use with that of its neighbors. The busier of the two nodes on each link sets the node weight for that link.

The detailed method used to determine the degree of node activity is presented in the "Compute Current Distances" routine of the appended simulation program. Once obtained, the level of node activity for each link is scaled linearly to fall in one of 128 "node weight bins". A pair of neighbor nodes which have no slot assignments (i.e. zero activity) will identify with bin number 1, while if one or both neighbors are saturated (as explained earlier) then the link between this pair of nodes identifies with node weight bin number 128.

Once a node weight bin is identified the actual node weight contained in that bin is added to the link weight to produce the total overall link distance which is then used by the Dijkstra dynamic routing event.

The node weight bin values may range in value from 0 in bin number 1 to 1024 in bin number 128. These bin values

are determined and assigned during program initialization according to input parameters which determine the "break point".

The break point is used to change the weighting of the node distance as the nodes become more active. See Figure 8. The break point consists of two coordinate

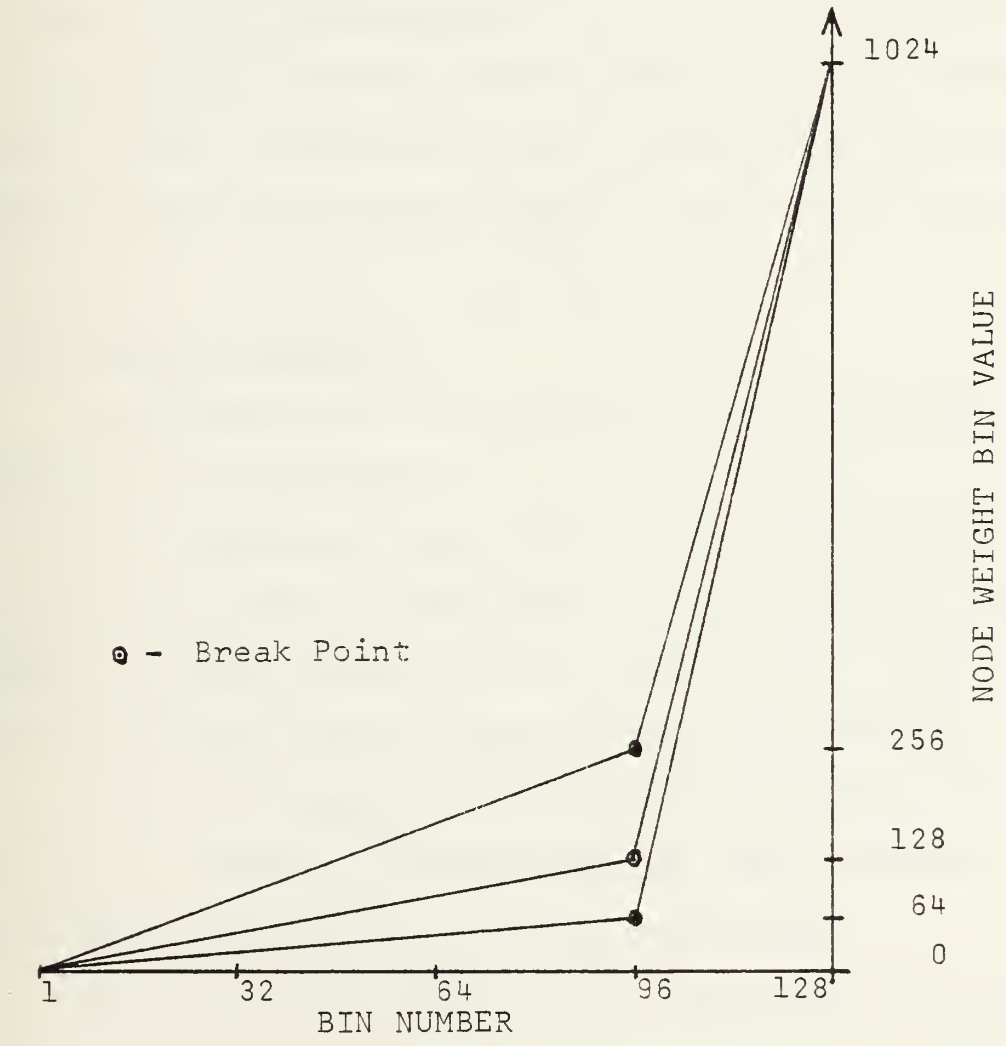


Figure 8 Node Weight Bin Values and the Break Points

parameters. The first coordinate identifies the bin and the second coordinate identifies the bin value at which the increment between adjacent bins changes. The use of the break point allows us to encourage the use of low activity nodes, and to discourage the use of nodes approaching saturation by assigning correspondingly low or high node weights.

Note that the bin values are actually assigned in monotonically increasing descrete increments. For example, use of the (96,256) break point results in an increment of 2.67 between each adjacent bin over bins 1 to 96. Bin 96 has a value of 256. The value of each successive bin is then incremented by 24.0 units of weight. Bin 128 has a value of 1024.

2. Program Parameters

The program was run using more than one hundred combinations of parameters.

All simulations were made with the same random number generator seed numbers. Therefore all simulations attempted to build the same virtual circuits, in the same order, and with the same time delay between circuit requirements.

The link weights were the same and constant for all simulations. However the node weights varied between simulations, depending on the break point used.

All simulations were run for 300 seconds of simulation time. There were no circuits in effect when each simulation began, and we observed that our network could accommodate

approximately twenty circuits when the mean call duration was 10 seconds. By 30 seconds into the simulation the network had attempted to establish approximately sixty circuits and had performed between six and fifteen best path update calculations. Accordingly, we presumed that the network reached its statistical steady-state operating condition by 30 seconds into the simulation. At this point in each simulation the appropriate counters were therefore re-initialized to remove the effect of the start up transient from the overall simulation statistics.

All time slots were 1 millisecond long and intermediate results were printed every 15 seconds. A much larger and more complete report was printed at the end of each simulation.

New virtual circuit requirements were generated according to an exponential distribution function with a mean value of 0.5 seconds. The simulation was 300 seconds long, and we observed that 590 virtual circuits were attempted during each simulation.

Virtual circuits, once established, remained in effect for a time duration also selected from an exponential distribution function. The mean value of this function was an input parameter. Three values were studied: 2, 5, and 10 seconds.

Three dynamic routing update periods were also studied. This parameter was assigned a value of 1, 3, or 5 seconds.

The receive signal stacking depth was assigned values between one and four.

Finally, three node weight break points were studied. These points were (96,64), (96,128), and (96,256).

V. CONCLUSIONS AND RECOMMENDATIONS FOR FURTHER STUDY

A. GENERAL

The time slot assignment algorithm was simulated both on a network employing dynamic (i.e. quasi-static) routing and on a network with static best path routing (that is, on a network where the best path neighbor assignments were held constant for the duration of the simulation). The static best path assignments were assigned manually and followed a least-hop routing strategy. All simulations, both static and dynamic, were made on the richly connected symmetric network presented earlier in Figure 1. It was not too difficult, due to the geometry of the network, to manually produce a static best path neighbor matrix which distributed the link and node usage approximately evenly over the network. The static best path neighbor assignments are contained in Appendix B. Virtual circuits built using static best path assignments were never longer than three hops, while some virtual circuits constructed during simulations employing dynamic routing were observed to make as many as seven hops, depending on the break point selected for the node weight portion of the distance calculation.

B. RESULTS AND OBSERVATIONS

1. General

Several tables of results are contained in Appendix C. The tables are crowded, but they are identical in format and the reader should have little difficulty reading them. In the discussion that follows we identify the general trends revealed in the simulation results.

a. Percentage of Circuits Established

It comes as no surprise that, when all of the other parameters are held constant, a greater percentage of calls can be established as we:

1) increase the allowable receive signal stacking depth,
2) decrease the mean duration of an established circuit, or
3) decrease the period between (i.e. increase the frequency of) the best path update cycles.

The results in Table C-1 show that decreasing the mean duration of a circuit has the greatest effect on the percentage of circuits that are established. Decreasing the average call duration from 10 seconds to 2 seconds generally results in a 30 to 70 percent improvement in the number of circuits established for both static and dynamic routing.

In the case of dynamic routing, we see that reducing the update period almost always results in a small (2 to 5 percent) improvement in the number of circuits established. This is because more frequent updates allow the heavily utilized nodes to be identified before they

reach saturation, so that future traffic may be routed through nodes with lower levels of utilization.

The data shows that increasing the slot stacking depth improves the percentage of circuits established. However, we note that the largest improvement with respect to this parameter is obtained by increasing the slot stacking depth from one to two. The number of circuits established generally continues to increase as the stacking depth is increased. However, the improvement is at a lower rate.

We see that as the ordinate of the break point is increased from 64 to 128 and then to 256, the percentage of established circuits tends to increase (when all other parameters are held constant). This can be explained by the fact that the (96,256) break point encourages the use of less busy nodes at the expense of using higher attenuation (i.e. higher energy) links. In contrast, the (96,64) break point appears to encourage the use of the lower attenuation links until the nodes on those links approach roughly 80 percent of saturation. The results in Table C-5 support this observation.

Table C-5 may also be used to explain why a greater percentage of circuits are established with static routing. The least-hop static routing uses all links approximately equally, regardless of the link attenuation or level of activity at the nodes on a path. The average energy for circuits built according to the static routing

scheme is almost always much higher than for the average circuit constructed with dynamic routing. Note also that a circuit built with the static least-hop routing strategy never uses more nodal assets (i.e. total slots) than does a dynamically routed circuit. Therefore, static least-hop routing conserves capacity throughout the network, and this, in turn, usually allows for a greater number of circuits to be active at any one time.

b. Average Number of Active Circuits

Table C-2 contains statistics concerning the average number of virtual circuits active at any one time during the simulation for the parameters shown. Trends in this table are difficult to identify, however, because the values in some of the columns are nearly identical.

If the average percentage of circuits established for one set of parameters is greater (or less) than the percentage established for another set of parameters, then we would expect that the average number of circuits active for that scheme should also be greater (or less) than the average number of circuits active for the other scheme. Thus we would expect that the values in this table should trend along the same lines as the values in Table C-1, and this is generally the case. For example, we see that increasing the slot stacking depth increases the average number of active circuits in proportion to the increase in the corresponding values in Table C-1.

A final point worth noting about the values in Table C-2 is that as the average call duration is decreased from 10 seconds to 2 seconds, the average number of circuits active at any one time decreases from approximately 13 to about 3.7. It is therefore not surprising that the shorter duration circuits are rebuffed less often: the network is very lightly loaded.

c. Average Number of Hops per Circuit

Table C-3 shows that the circuits established with static least-hop routing make fewer hops than the dynamically routed circuits. This is just as it should be. A more subtle trend revealed by these figures is that the variation of any parameter which generally increases the percentage of circuits established (i.e. decreasing the mean circuit duration or update period, or increasing the stacking depth) generally causes an increase in the average number of hops. This tells us that the additional circuits are, on the average, following longer paths.

We also note that increasing the ordinate of the break point tends to reduce the average number of hops per circuit. As the ordinate is increased the dynamic routing scheme appraoches the least-hop routing strategy. Similarly, as the ordinate is decreased the dynamic routing scheme tends toward the least energy routing strategy. This is verified by the data in Table C-5.

d. Largest Number of Hops

Table C-4 lists the number of circuits that made the largest number of hops for any combination of the parameters studied. Of the 518 circuit requirements entered into the network between the time the counters were reset at 30 seconds into the simulation, and the end of the simulation 270 seconds later, we see that for static routing, anywhere from less than one tenth to nearly one fifth of the established circuits took three hops. These figures again illustrate that the longer multi-hop messages are more likely to be established under the lightly loaded network condition (i.e., when the mean circuit duration is 2 seconds).

Three sets of dynamic routing parameters caused one of the 518 circuits to be established over a path seven hops long. We were concerned that the use of the (96,64) break point might so bias the distance function and best path calculation in favor of the low attenuation links, that circuits would make an inordinate number of hops. However, the data does not support this concern.

e. Average Energy per Circuit

The "energy factors" presented in Table C-5 are our own convention. We derived, from the link attenuation value for each link, a representative figure for the energy required for communications over that link. The simulation program kept track of which circuits were built and which links were used. At the end of the simulation, the average

energy per established circuit was divided by 100000 to produce the "energy factor" which is displayed in Table C-5 for each set of parameters.

We see that the (96,64) break point definitely results in preferential use of the lower energy links, and that increasing the break point ordinate results in an increase of the average energy factor.

2. Summary

In summary, for the parameters that were studied, the average virtual circuit duration has the greatest effect on the overall statistics. The update period, coordinates of the break point, and slot stacking depth generally have a smaller impact on the statistics. The effect of increasing the stacking depth tends to be reduced as the stacking depth is increased. If we seek to limit the overall radiated energy of the network, then dynamic routing (with a low break point such as (96,64)) should be used. However, if maximum throughput is required and we can afford to suffer the consequences of increased signal energy, then our results suggest that users should keep calls as brief as possible and that, in our test network, either the static least-hop or dynamic routing, with a (96,256) break point, should be used. The major conclusion of this report is that it is possible to route in a way that reduces the average energy transmitted per message without substantially decreasing the network throughput.

C. RECOMMENDATIONS FOR FURTHER STUDY

During the development and analysis of the proposed packet-switched network time slot assignment algorithm, it became apparent that there were several courses that future research could follow. Listed below, in no particular order, are several recommendations for further study. Some are mere enhancements to the appended simulation program while others would require the generation of new programs, or the integration of two or more of the simulation programs developed by previous NPS graduate students.

We know that there are several ways to calculate the link distances. Our distance calculations were a function of both path attenuation and node utilization. The node utilization calculation was based entirely on the mutual availability of slots remaining between each pair of directly connected nodes, as a result of the slot assignments for virtual circuits already active between that pair of nodes. We assumed that data message packets could always be stored in a queue at each node and forwarded as slots became available. Therefore we did not simulate or study the actual performance of our algorithm with respect to data traffic. If future studies simulate the processing of both data and virtual circuit voice traffic, then it seems desirable to include the data queue size and/or data packet message delay as elements in the distance calculation.

It is sensible to expect that some percentage of the callers whose initial (and subsequent) calls were rebuffed might attempt to re-dial the same call at some later time. It would not be difficult to modify the existing simulation program to accommodate this activity; the results might be very interesting.

Future studies might examine other routing algorithms and/or simulate the actual transmission and handling of update messages used to carry the distance information from node to node throughout the network. Along these lines, it might be worthwhile to combine our slot assignment scheme with Heritsch's [Ref. 9] hierarchical routing protocol.

The slot assignment algorithm should be tested on a larger network. Several possibilities come to mind. It seems reasonable to exploit the previous research of Bond [Ref. 3] and Kane [Ref. 4] for this. Their work concentrated on a prototype packet radio network (for a MAB) composed of approximately seventy-five nodes. A network this large might require a prohibitive amount of computer execution time to simulate adequately, but their work nonetheless provides a good starting point for the study of larger tactical networks.

We have allowed all of the nodes in our network to originate and receive voice traffic equally. The nodes in an actual tactical packet radio network would generate varying amounts of voice and data traffic, and the addressees for this traffic would not be uniformly distributed across

all net members. In a fast moving tactical situation most of the network traffic would be command and fire support coordination type traffic, while the predominant type of traffic between battles would be more administrative and logistical in nature. Bond's work [Ref. 3] provides statistics concerning the type (data or voice) of traffic the different nodes in a MAB have generated historically.

Future studies could include the effects of terrain on network connectivity and link attenuations as originally studied by Kane [Ref. 4]. The STAR Terrain Model would be useful for the purpose and also for the simulated movement of nodes from position to position across STAR's simulated battlefield.

None of the previously mentioned and referenced research at NPS has provided more than a cursory analysis and discussion of some of the most difficult aspects of an actual packet radio network implementation. Briefly these aspects include, but are not limited to:

1) Initializing and starting the network in operation.

2) The effects of changes in network topology caused by broken links or by nodes joining or leaving the network.

3) Identification and use of alternate or "next best path" routes to increase network throughput.

It should be instructive to vary parameters such as the number of time slots per frame, the time slot duration, the update period or the coordinates of the "break point", etc.,

in the existing time slot assignment algorithm and study the effect on network performance.

In conslusion, this thesis was a preliminary investigation of a proposed time slot assignment algorithm. We recognize that our algorithm is but one of several possible schemes. We have identified its broad performance characteristics and know that the algorithm works. We believe that the concept of implementing a future military packet radio network with integrated voice and data traffic utilizing spread spectrum and CDMA techniques in conjunction with some type of TDMA time slot assignment scheme is a viable notion worthy of further study.

APPENDIX A

LINK ATTENUATIONS (in dB)

FROM NODE \ TO NODE	1	2	3	4	5	6	7
1		119.7	91.3	133.2	127.6		
2	119.7		106.5			81.3	
3	91.3	106.5		92.9		101.9	94.0
4	133.2		92.9		121.8		122.4
5	127.6			121.8			
6		81.3	101.9				103.7
7			94.0	122.4		103.7	
8				97.8	111.1		105.5
9		122.2				97.6	
10						113.2	98.6
11							81.1
12					133.3		
13							

FROM NODE \ TO NODE	8	9	10	11	12	13
1						
2		122.2				
3						
4	97.8					
5	111.1				133.3	
6		97.6	113.2			
7	105.5		98.6	81.1		
8				110.9	130.0	
9			123.4			117.6
10		123.4		131.2		118.6
11	110.9		131.2		100.6	123.3
12	130.0			100.6		140.7
13		117.6	118.6	123.3	140.7	

APPENDIX B

STATIC BEST PATH NEIGHBOR ASSIGNMENTS

FROM \ TO	1	2	3	4	5	6	7	8	9	10	11	12	13
1	-	2	3	4	5	3	3	4	2	2	5	5	2
2	1	-	3	3	1	6	6	1	9	6	9	9	9
3	1	2	-	4	1	6	7	4	2	6	7	4	6
4	1	1	3	-	5	3	7	8	3	7	8	5	8
5	1	1	4	4	-	1	4	8	1	12	8	12	12
6	2	2	3	3	3	-	7	7	9	10	10	10	9
7	4	3	3	4	8	6	-	8	6	10	11	11	10
8	5	4	4	4	5	7	7	-	11	11	11	12	12
9	2	2	6	2	13	6	10	13	-	10	10	13	13
10	6	9	6	7	11	6	7	11	9	-	11	13	13
11	8	10	7	8	12	10	7	8	13	10	-	12	13
12	5	5	5	8	5	13	8	8	13	11	11	-	13
13	12	9	9	12	12	10	11	11	9	10	11	12	-

APPENDIX C

RESULTS OF THE SIMULATION

TABLE C-1 PERCENTAGE OF CIRCUITS ESTABLISHED

			Average Virtual Circuit Duration								
			2 sec			5 sec			10 sec		
		Update Period	1	3	5	1	3	5	1	3	5
		Slot Depth									
Static Routing		1		97.1			80.9			60.0	
		2		99.6			90.3			70.8	
		3		99.4			94.4			72.8	
		4		99.4			93.2			76.1	
Dynamic Routing	(96, 64)	1	90.5	91.9	88.6	76.1	72.6	71.0	55.8	51.2	52.1
		2	98.3	97.1	95.4	84.7	84.4	81.9	66.6	63.5	63.1
		3	99.0	96.9	95.9	92.9	81.0	86.5	70.8	70.8	66.8
		4	99.0	99.4	97.1	94.6	87.5	84.2	71.0	73.2	70.1
	(96, 128)	1	95.0	93.2	91.3	75.5	75.7	75.5	60.2	56.8	53.3
		2	99.2	97.9	96.7	90.9	88.4	87.3	70.3	71.0	68.1
		3	99.4	99.2	98.1	94.6	91.3	89.2	70.1	73.9	74.5
		4	99.4	98.1	97.5	97.7	92.7	90.9	72.2	71.6	74.5
	(96, 256)	1	97.5	91.9	91.3	81.1	74.9	76.6	55.0	54.8	55.4
		2	99.8	96.1	96.7	94.6	92.7	89.8	73.7	72.2	67.2
		3	98.8	99.2	97.7	95.8	95.0	94.4	76.8	74.3	75.9
		4	99.8	99.4	98.6	97.1	93.8	92.3	79.3	77.0	76.6

TABLE C-2 AVERAGE NUMBER OF VIRTUAL CIRCUITS ACTIVE AT ANY ONE TIME

			Average Virtual Circuit Duration								
			2 sec			5 sec			10 sec		
	Update Period		1	3	5	1	3	5	1	3	5
		Slot Depth									
Static Routing		1		3.7			7.8			11.5	
		2		3.8			8.7			13.5	
		3		3.8			9.0			13.9	
		4		3.8			8.9			14.7	
Dynamic Routing	(96, 64)	1	3.4	3.5	3.4	7.3	7.0	6.8	10.6	9.9	10.0
		2	3.7	3.7	3.6	8.2	8.1	7.9	12.7	12.0	12.0
		3	3.8	3.7	3.6	8.9	7.8	8.3	13.3	13.3	12.7
		4	3.8	3.8	3.7	9.1	8.4	8.1	13.5	13.7	13.3
	(96, 128)	1	3.6	3.5	3.4	7.2	7.3	7.2	11.4	10.8	10.2
		2	3.8	3.7	3.6	8.7	8.5	8.4	13.3	13.1	13.0
		3	3.8	3.8	3.7	9.1	8.8	8.6	13.4	14.1	14.3
		4	3.8	3.7	3.7	9.4	8.9	8.7	13.9	13.8	14.2
	(96, 256)	1	3.7	3.4	3.4	7.8	7.2	7.3	10.6	10.4	10.6
		2	3.8	3.6	3.7	9.0	8.9	8.6	14.2	13.7	12.8
		3	3.8	3.8	3.7	9.2	9.1	9.0	15.0	14.1	14.4
		4	3.8	3.8	3.7	9.3	9.0	8.8	15.4	14.9	14.8

TABLE C-3 AVERAGE NUMBER OF HOPS PER ESTABLISHED VIRTUAL CIRCUIT

			Average Virtual Circuit Duration								
			2 sec			5 sec			10 sec		
Update Period			1	3	5	1	3	5	1	3	5
		Slot Depth									
Static Routing		1		1.83			1.76			1.68	
		2		1.84			1.80			1.78	
		3		1.83			1.81			1.73	
		4		1.83			1.83			1.79	
Dynamic Routing	(96, 64)	1	2.15	2.22	2.20	2.15	2.13	2.05	2.16	2.09	2.03
		2	2.23	2.20	2.17	2.17	2.15	2.16	2.23	2.20	2.07
		3	2.19	2.14	2.19	2.15	2.24	2.15	2.18	2.15	2.09
		4	2.23	2.23	2.20	2.23	2.18	2.14	2.26	2.17	2.09
	(96, 128)	1	2.18	2.13	2.13	2.05	2.06	1.95	1.98	1.94	1.99
		2	2.13	2.11	2.15	2.08	2.05	2.05	2.09	2.02	1.98
		3	2.18	2.13	2.13	2.05	2.05	2.04	2.05	2.11	1.95
		4	2.18	2.19	2.08	2.02	2.00	2.00	2.08	2.09	2.01
	(96, 256)	1	2.10	2.15	2.12	2.01	1.97	1.88	1.91	1.86	1.86
		2	2.11	2.10	2.17	1.96	1.91	1.92	1.98	1.91	1.89
		3	2.09	2.15	2.09	1.95	1.97	1.98	1.98	2.00	1.94
		4	2.15	2.08	2.09	1.95	1.96	1.98	1.98	1.94	1.92

TABLE C-4 LARGEST NUMBER OF HOPS

(Largest Nr. of Hops / Nr. of Circuits Making Largest Nr. of Hops)

			Average Virtual Circuit Duration								
			2 sec			5 sec			10 sec		
		Update Period	1	3	5	1	3	5	1	3	5
		Slot Depth									
Static Routing		1		3/98			3/72			3/46	
		2		3/103			3/89			3/64	
		3		3/102			3/92			3/61	
		4		3/102			3/97			3/73	
Dynamic Routing	(96, 64)	1	5/3	5/9	5/9	6/1	5/12	5/4	5/9	6/2	5/5
		2	5/7	5/5	5/3	5/6	6/1	5/7	7/1	5/9	7/1
		3	5/2	5/6	5/1	5/3	6/1	5/9	5/8	6/3	5/1
		4	5/5	6/1	5/5	6/1	6/3	6/1	6/1	6/1	6/1
	(96, 128)	1	5/7	5/6	5/2	5/7	5/4	6/1	5/2	5/6	5/3
		2	5/3	5/4	6/1	5/4	5/5	5/6	5/7	5/1	5/3
		3	5/4	6/1	6/1	5/1	5/4	5/4	5/3	6/1	5/4
		4	5/4	6/2	5/3	4/27	6/1	5/2	6/1	5/4	5/3
	(96, 256)	1	6/1	6/1	5/6	6/1	5/1	5/1	4/14	5/2	5/1
		2	6/1	5/10	5/7	4/27	5/2	4/20	5/1	5/2	4/12
		3	6/1	7/1	5/6	4/27	5/1	4/27	5/2	5/2	5/1
		4	6/1	5/11	5/6	5/2	5/2	5/2	4/17	4/15	5/1

TABLE C-5 AVERAGE ENERGY FACTOR PER ESTABLISHED VIRTUAL CIRCUIT

			Average Virtual Circuit Duration								
			2 sec			5 sec			10 sec		
		Update Period	1	3	5	1	3	5	1	3	5
		Slot Depth									
Static Routing		1		1.51			1.38			1.37	
		2		1.51			1.54			1.48	
		3		1.51			1.46			1.42	
		4		1.53			1.51			1.55	
Dynamic Routing	(96, 64)	1	0.15	0.21	0.12	0.49	0.60	0.47	1.21	0.84	1.06
		2	0.11	0.15	0.21	0.59	0.53	0.55	1.07	1.10	0.79
		3	0.17	0.19	0.13	0.66	0.75	0.57	1.03	1.04	1.03
		4	0.11	0.13	0.11	0.60	0.63	0.32	0.87	1.13	1.02
	(96, 128)	1	0.46	0.39	0.43	1.16	0.98	0.65	1.09	1.37	1.09
		2	0.33	0.44	0.37	1.07	0.72	0.78	1.27	1.10	1.26
		3	0.20	0.39	0.29	0.76	0.68	0.69	1.26	1.25	1.11
		4	0.22	0.30	0.42	0.74	0.60	0.74	1.07	1.21	0.96
	(96, 256)	1	0.72	0.70	0.63	1.11	1.14	0.97	1.17	1.26	1.03
		2	0.56	0.66	0.61	1.02	1.08	1.07	1.21	1.27	1.31
		3	0.52	0.63	0.72	1.04	0.94	0.88	1.22	1.43	1.19
		4	0.51	0.62	0.79	0.89	0.92	0.98	1.15	1.20	1.15

```
FILE: THESIS   SIMS      A1  NAVAL POSTGRADUATE SCHOOL


//TRIC1966 JOB (1966,0132),'TRITCHLER 1642',CLASS=C
//*MAIN ORG=NPGVM1.1966P,LINES=(6)
//*FORMAT PR,DDNAME=,DEST=LOCAL
// EXEC SIM25C
//SYSPRINT DD SYSOUT=A
//SIM.SYSLIN DD UNIT=3330V,MSVGP=PUB4B,DISP=(OLD,KEEP),
//     DSN=MSS.S1966.THESIX.LOADLIB
//SIM.SYSIN DD *
PREAMBLE
''
NORMALLY MODE IS INTEGER
''
PERMANENT ENTITIES
  EVERY NODE HAS A TRANSMIT.PERCENT, A RECEIVE.PERCENT, A GROUP AND
    A FAMILY
  DEFINE TRANSMIT.PERCENT AND RECEIVE.PERCENT AS REAL VARIABLES
''
GENERATE LIST ROUTINES
''
TEMPORARY ENTITIES
  EVERY MESSAGE HAS A CKT.NR, A TYPE, AN ORIGINATOR, A DESTINATION,
    A FM.NODE, A TO.NODE, A START.TIME, A HOP.COUNT, A SLOT.ARRIVAL,
    A SLOT.ASSIGN, A RECSLOT, A DIRECTION, A CUM.ENERGY, A INFO1,
    A INFO2, A INFO3, A INFO4, A INFO5, A INFO6, A INFO7, A INFO8 AND
    A INFO9
    DEFINE START.TIME, HOP.COUNT AND CUM.ENERGY AS REAL VARIABLES
''
EVENT NOTICES INCLUDE STOP.SIMULATION, NEW.CKT.REQMT,
  INITIAL.REQ.FOR.SVC, RESPONSE.REQ.FOR.SVC, FINAL.ASSIGNMENT.NOTICE,
  UPSTREAM.BREAK.DOWN, DOWNSTREAM.BREAK.DOWN, DIJK.MANIPULATION AND
  RE.MOVE.TRANSIENT.EFFECT
  EVERY INITIAL.REQ.FOR.SVC HAS A SVC1.MSG
  EVERY RESPONSE.REQ.FOR.SVC HAS A SVC2.MSG
  EVERY FINAL.ASSIGNMENT.NOTICE HAS A SVC3.MSG
  EVERY UPSTREAM.BREAK.DOWN HAS A U.B.D.MSG
  EVERY DOWNSTREAM.BREAK.DOWN HAS A D.B.D.MSG
''
PRIORITY ORDER IS UPSTREAM.BREAK.DOWN, DOWNSTREAM.BREAK.DOWN,
  STOP.SIMULATION, RE.MOVE.TRANSIENT.EFFECT AND DIJK.MANIPULATION
''
ACCUMULATE CUM.MEAN AS THE MEAN, CUM.VARIANCE AS THE VARIANCE,
CUM.STD.DEVIATION AS THE STD.DEV, MAX.ACTIVE AS THE MAXIMUM,
MIN.ACTIVE AS THE MINIMUM OF ACTIVE
''
DEFINE HOUSEKEEPING AS A RELEASABLE ROUTINE
DEFINE ECHO.PRINT.INPUT.DATA AS A RELEASABLE ROUTINE
''
DEFINE USE AS A 3-DIMENSIONAL INTEGER ARRAY
DEFINE TSLT AS A 1-DIMENSIONAL INTEGER ARRAY
DEFINE BEST.PATH AS A 2-DIMENSIONAL INTEGER ARRAY
DEFINE FAM.OF.GRP AS A 1-DIMENSIONAL INTEGER ARRAY
DEFINE LINKABLE AS A 2-DIMENSIONAL INTEGER ARRAY
DEFINE NODE.COUNT AS A 2-DIMENSIONAL INTEGER ARRAY
DEFINE DIJKSTRA AS A 2-DIMENSIONAL REAL ARRAY
DEFINE DISTANCE AS A 2-DIMENSIONAL REAL ARRAY
DEFINE ATTENUATION AS A 2-DIMENSIONAL REAL ARRAY
DEFINE PATH.AVAIL AS A 2-DIMENSIONAL INTEGER ARRAY
DEFINE NODE.SCALE AS A 1-DIMENSIONAL REAL ARRAY
DEFINE LINK.WEIGHT AS A 2-DIMENSIONAL REAL ARRAY
DEFINE ENERGY AS A 2-DIMENSIONAL REAL ARRAY
DEFINE NERGY AS A 2-DIMENSIONAL REAL ARRAY
DEFINE LIN.K.USED AS A 1-DIMENSIONAL INTEGER ARRAY
DEFINE LI.NK.NR AS A 2-DIMENSIONAL INTEGER ARRAY
DEFINE UPDATE TO MEAN 1
DEFINE PACKET TO MEAN 2
DEFINE PARTIAL.BREAKDOWN TO MEAN 3
DEFINE FULL.BREAKDOWN TO MEAN 4
DEFINE REMOVE.LOOP TO MEAN 5
DEFINE MAX.SLOT.DEPTH, STARTING.MAX.SLOT.DEPTH AND ENDING.MAX.SLOT.DEPTH
  AS INTEGER VARIABLES
DEFINE REPORT.COUNTER AS AN INTEGER VARIABLE
DEFINE CKT.TOTAL, CKT.ESTAB, CKT.FAILED, CKT.SUM,
```

```
   AND CKT.DISESTAB AS INTEGER VARIABLES
DEFINE UP.ROUTE, DOWN.ROUTE AND ACTIVE AS INTEGER VARIABLES
DEFINE TRNS.PCNT AND RCV.PCNT AS REAL VARIABLES
DEFINE GRPS, FMLYS AND NGFS AS INTEGER VARIABLES
DEFINE SPECIFY.OUTPUT, PRNT, PRT AND LTD.PRINT AS INTEGER VARIABLES
DEFINE TEST.DURATION, SLOT.DURATION, PROCESSING.TIME, PROP.DELAY.TIME,
   MEAN.CKT.ESTAB AND MEAN.DURATION.OF.CKT AS REAL VARIABLES
DEFINE NODAL.MEAN.CKT.ESTAB AS A REAL VARIABLE
DEFINE SLOTS, LINKS AND MAX.LINKS.PER.NODE AS INTEGER VARIABLES
DEFINE LINK.NODE.RATIO AS A REAL VARIABLE
DEFINE STARTER AS AN INTEGER VARIABLE
DEFINE IN.GROUP AND IN.FAMILY AS REAL VARIABLES
DEFINE LONG.TIME.EST, AVG.P.BD, LONG.P.BD, AVG.C.BD, LONG.C.BD AND
   AVG.TIME.EST AS REAL VARIABLES
DEFINE DELAY.SUM, SUM.DURATION, AND AVG.DURATION
   AS REAL VARIABLES
DEFINE CKT.GREATEST AND CKT.LONG.TIME.EST AS INTEGER VARIABLES
DEFINE MAX.CKTS.IN.SIM AS AN INTEGER VARIABLE
DEFINE HOP.GREATEST, HOP.SUM AND HOP.AVG AS REAL VARIABLES
DEFINE TOT.HOP.GREATEST AS AN INTEGER VARIABLE
DEFINE SUM.BD.TIME.ALL.CKT, AVG.BD.TIME, TOT.P.BD AND TOT.C.BD AS
   REAL VARIABLES
DEFINE P.BD.COUNTER AND C.BD.COUNTER AS INTEGER VARIABLES
DEFINE CKTS.BD AS AN INTEGER VARIABLE
DEFINE UP.DATE.PERIOD AND RE.PORT.PERIOD AS REAL VARIABLES
DEFINE CHANGE.FLAG AS AN INTEGER VARIABLE
DEFINE TOT.DIJK.CALLED, BACKTRACK.OR.LOOPBACK AND ACT.LOOP.REMOVE AS
   INTEGER VARIABLES
DEFINE THEO.CAP AS A REAL VARIABLE
DEFINE BRK.X.POINT AND BRK.Y.POINT AS INTEGER VARIABLES
DEFINE NODE.MAX.SCALE.WEIGHT AS A REAL VARIABLE
DEFINE ROUTING.ALGORITHM.SELECTOR AS AN INTEGER VARIABLE
DEFINE FRACT.OF.SUCCESSFUL.CALLS AND AVG.ACTIVE AS REAL VARIABLES
DEFINE E.SUM AND E.SUB.K.BAR AS REAL VARIABLES
END ''OF PREAMBLE
''
''    THIS IS THE MAIN PROGRAM
''
MAIN
''
LET LINES.V = 82
DEFINE TRANSIENT.TIME AS A REAL VARIABLE
START NEW PAGE
PRINT 3 LINES AS FOLLOWS
      PROGRAM TO INVESTIGATE THE EFFECTS OF STACKING RECEIVE SIGNALS TO
         VARIOUS DEPTHS IN TIME SLOTS.
   ------------------------------------------------------------------
SKIP 2 OUTPUT LINES
''
''    THE MAIN PROGRAM CALLS THE HOUSEKEEPING ROUTINE THAT SETS THE
''       THE VALUE OF ALL INPUT VARIABLES THAT REMAIN CONSTANT FOR ALL
''       RUNS OF THE SIMULATIONS.  THIS ALLOWS THE MAIN PROGRAM TO ACT AS
''       THE DRIVER ROUTINE FOR THE SIMULATION.  THE MAIN PROGRAM CAN BE
''       STRUCTURED TO CHANGE CERTAIN CONDITIONS OF THE SIMULATION AND
''       THEN RERUN THE SIMULATION AGAIN.
''
PERFORM HOUSEKEEPING
RELEASE HOUSEKEEPING
RELEASE ECHO.PRINT.INPUT.DATA
''
'DO.IT.AGAIN'
''
''    TEST TO SEE IF THE ENTIRE SIMULATION IS COMPLETE.
''
IF MAX.SLOT.DEPTH GT ENDING.MAX.SLOT.DEPTH
   GO TO FINISH
ALWAYS
''
''    INITIALIZE IMPORTANT COUNTING VARIABLES AND ARRAYS FOR EACH ITERA-
''       TION OF THE SIMULATION.
''
LET TIME.V = 0.000000000
```

```
''
IF ROUTING.ALGORITHM.SELECTOR EQ 1
   RELEASE BEST.PATH(*,*)
   PERFORM ARRAY.INITIALIZATION
ALWAYS
''
RESET TOTALS OF ACTIVE
''
RESERVE LIN.K.USED(*) AS LINKS
LET REPORT.COUNTER = 0
LET CKT.TOTAL = 0
LET CKT.SUM  = 0
LET CKT.ESTAB = 0
LET CKT.FAILED = 0
LET CKT.DISESTAB = 0
LET UP.ROUTE = 0
LET DOWN.ROUTE = 0
LET ACTIVE = 0
LET HOP.SUM = 0.
LET HOP.GREATEST = 0.
LET TOT.HOP.GREATEST  = 0
LET HOP.AVG = 0.
LET DELAY.SUM = 0.
LET DURATION = 0.
LET SUM.DURATION = 0.
LET AVG.DURATION = 0.
LET LONG.TIME.EST = 0.
LET AVG.TIME.EST = 0.
LET AVG.P.BD = 0.
LET LONG.P.BD = 0.
LET AVG.C.BD = 0.
LET LONG.C.BD = 0.
LET CKT.LONG.TIME.EST  = 0
LET AVG.BD.TIME = 0.
LET SUM.BD.TIME.ALL.CKT = 0.
LET CKTS.BD = 0
LET P.BD.COUNTER = 0
LET C.BD.COUNTER = 0
LET TOT.P.BD = 0.
LET TOT.C.BD = 0.
LET CHANGE.FLAG = 1
LET TOT.DIJK.CALLED = 0
LET BACKTRACK.OR.LOOPBACK = 0
LET ACT.LOOP.REMOVE = 0
LET F.SUM = 0.0
LET F.SUB.K.BAR = 0.0
LET FRACT.OF.SUCCESSFUL.CALLS = 0.0
LET AVG.ACTIVE = 0.0
''
''    RELEASE THE SYSTEM'S "SEED.V" ARRAY, THEN RE-DIMENSION THIS ARRAY
''       AND READ IN THE SAME SET OF RANDOM NUMBER SEEDS FOR EACH ITERA-
''       TION OF THE SIMULATION.
''
RELEASE SEED.V(*)
RESERVE SEED.V(*) AS 10
READ SEED.V
''
''    CALCULATE THE THEORETICAL ABSOLUTE MAXIMUM CAPACITY FOR A RICHLY
''       CONNECTED NETWORK.
''
LET NR.XMIT.SLOTS = TRUNC.F(REAL.F(SLOTS) / (1.0 + 1.0 /
  REAL.F(MAX.SLOT.DEPTH)))
LET THEO.CAP = REAL.F(N.NODE) * REAL.F(NR.XMIT.SLOTS)
''
RESERVE USE(*,*,*) AS N.NODE BY SLOTS BY 6
START NEW PAGE
PRINT 7 LINES WITH MAX.SLOT.DEPTH AS FOLLOWS
XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
XX                                                                     XX
XX                        RESULTS OF SIMULATION                        XX
XX                                 FOR                                 XX
XX                       MAXIMUM SLOT DEPTH = **                       XX
```

```
XX                                                                   XX
XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
SKIP 2 OUTPUT LINES
''
''   SCHEDULE INITIAL EVENTS
''
IF ROUTING.ALGORITHM.SELECTOR EQ 1
  SCHEDULE A CIJK.MANIPULATION AT 0.C0000000
ALWAYS
SCHEDULE A STOP.SIMULATION IN RE.PORT.PERIOD UNITS
SCHEDULE A NEW.CKT.REQMT IN EXPONENTIAL.F(MEAN.CKT.ESTAB,1) UNITS
LET TRANSIENT.TIME = 30.000
SCHEDULE A RE.MOVE.TRANSIENT.EFFECT IN TRANSIENT.TIME UNITS
''
START SIMULATION
''
RELEASE USE(*,*,*)
RELEASE DIJKSTRA(*,*)
RELEASE DISTANCE(*,*)
RELEASE PATH.AVAIL(*,*)
RELEASE NODE.SCALE(*)
RELEASE LIN.K.USED(*)
IF ROUTING.ALGORITHM.SELECTOR EQ 1
  RELEASE BEST.PATH(*,*)
ALWAYS
''
''   RUN THE SIMULATION AGAIN FOR A NEW SLOT DEPTH
''
LET MAX.SLOT.DEPTH = MAX.SLOT.DEPTH + 1
GO TO DO.IT.AGAIN
''
'FINISH'
SKIP 3 OUTPUT LINES
PRINT 2 LINES AS FOLLOWS
          TOTAL, COMPLETE, AND ABSOLUTE END OF THE SIMULATION.
          ---------------------------------------------------------
STOP
END ''OF MAIN
''
''   THIS ROUTINE READS IN ALL OF THE VARIABLES IN THE SIMULATION.
''     BY PROPER STRUCTURING OF THIS ROUTINE AND THE "MAIN" PROGRAM,
''     THE SIMULATION CAN BE MADE TO SUCCESSIVELY RERUN ITSELF USING
''     ANY NUMBER OF NEW INPUT PARAMETERS ON EACH RUN.
''
ROUTINE FOR HOUSEKEEPING
''
DEFINE ADJUSTED.ATT, EN.ERGY AND WT AS REAL VARIABLES
''
''   SPECIFY.OUTPUT IS AN INTEGER WHICH, IN PART, CONTROLS THE QUANTITY
''     AND TYPE OF PRINTED OUTPUT.
''       0 =>  ALL INPUT DATA AND THE QUARTERLY RESULTS OF THE SIMULA-
''               TION ARE OUTPUT.  THIS IS THE NORMAL OUTPUT MODE.
''       1 =>  ONLY THE INPUT DATA AND THE DATA SPECIFIED BY THE PRO-
''               GRAMMER IN "SPECIAL.OUTPUT" ARE PRINTED OUT.  QUARTERLY
''               RESULTS OF THE SIMULATION ARE NOT PRINTED.
''       2 =>  ONLY THE DATA SPECIFIED IN "SPECIAL.OUTPUT" IS OUTPUT.
''
READ SPECIFY.OUTPUT
''
''   PRNT IS AN INPUT VARIABLE THAT CONTROLS THE AMOUNT OF DIAGNOSTIC
''     PRINTING ASSOCIATED WITH BUILDING AND DISESTABLISHING VIRTUAL
''     CIRCUITS.
''       0 ==>  ANNOUNCES EACH NEW CIRCUIT REQUIREMENT AND WHETHER THE
''                CIRCUIT IS EVENTUALLY ESTABLISHED OR BROKEN DOWN BE-
''                CAUSE SLOTS WERE NOT AVAILABLE AT ONE OF THE NODES
''                ALONG THE PATH.
''       1 ==>  0 + PRINTS THE SLOT ASSIGNMENTS AT EACH NODE AFTER THE
''                FIRST QUARTER AND AT THE END OF EACH RUN OF THE SIMU-
''                LATION.
''       2 ==>  1 + SELECTIVE PRINTING OF OTHER INFORMATION.
''       4 ==>  SUPPRESSES THE ABOVE LISTED DIAGNOSTIC PRINTING.
''
```

```
''   CAUTION:  AS AN AID TO DE-BUGGING THE PROGRAM, THE VALUE OF PRNT
''     MAY BE CHANGED BY THE PROGRAM SEVERAL TIMES DURING EXECUTION.
''
READ PRNT
''
''   PRT IS AN INPUT VARIABLE THAT CONTROLS THE AMOUNT OF DIAGNOSTIC
''     PRINTING ASSOCIATED WITH THE DYNAMIC ROUTING RELATED OPERATIONS
''     OF THE PROGRAM.
''
READ PRT
''
''   LTD.PRINT IS ANOTHER INPUT VARIABLE THAT WAS ADDED AT THE LAST MIN-
''     UTE TO LIMIT THE VOLUME OF PRINTED OUTPUT IN THE PERIODIC REPORTS
''     PRODUCED IN THE STOP.SIMULATION EVENT.
''       0 ==>  ALL OF THE REGULAR OUTPUT IS PRODUCED AS DETERMINED BY
''                THE SPECIAL.OUTPUT, PRNT, AND PRT VARIABLES EXPLAINED
''                ABOVE.
''       1 ==>  THE VOLUME OF PRINTED OUTPUT IS LIMITED.
''
READ LTD.PRINT
''
''   READ THE "ROUTING.ALGORITHM.SELECTOR" WHICH IS USED TO IDENTIFY
''     WHICH TYPE OF ALGORITHM THE SIMULATION WILL SIMULATE.
''       1 ==>  DYNAMIC ROUTING ACCORDING TO THE DIJKSTRA ALGORITHM.
''       2 ==>  STATIC BEST PATH LEAST HOP ROUTING.
''
READ ROUTING.ALGORITHM.SELECTOR
''
''   READ THE NUMBER OF NODES IN THE NETWORK AND THE NODAL TRANSMIT AND
''     RECEIVE FACTORS.
''
READ N.NODE
IF SPECIFY.OUTPUT LE 1
  PRINT 2 LINES AS FOLLOWS
NODE      TRANSMIT      RECEIVE      GROUP      FAMILY
 NO.        FACTOR       FACTOR      (PGM #)    (PGM #)
REGARDLESS
''
CREATE EVERY NODE
FOR EVERY NODE
  READ TRANSMIT.PERCENT(NODE), RECEIVE.PERCENT(NODE), GROUP(NODE) AND
     FAMILY(NODE)
''
''   TRNS.PCNT AND RCV.PCNT ARE THE SUM OF TRANSMIT AND RECEIVE FACTORS.
''     GROUP NUMBERS ARE ADDED TO N.NODE TO GET PROGRAM GROUP NUMBERS.
''     FAMILY NUMBERS ARE ADDED TO N.NODE + THE HIGHEST GROUP NUMBER TO
''     GET THE PROGRAM FAMILY NUMBERS.  WITHIN THE SIMULATION, GROUPS
''     AND FAMILIES ARE HANDLED AS IF THEY WERE SUPER-NODES.  A USEFUL
''     ANALOGY WOULD BE TO ENVISION MANY SUB-NODES WITHIN A GROUP OR
''     FAMILY SUPER-NODE.  ACCESS TO THE SUB-NODES IS CONTROLLED BY THE
''     SUPER-NODE'S "ADDRESS".
''
FOR I = 1 TO N.NODE, DO
  LET TRNS.PCNT = TRNS.PCNT + TRANSMIT.PERCENT(I)
  LET RCV.PCNT = RCV.PCNT + RECEIVE.PERCENT(I)
  IF GRPS LT GROUP(I)
     LET GRPS = GROUP(I)
  REGARDLESS
''
''   SET PROGRAM GRP NUM
''
  LET GROUP(I) = GROUP(I) + N.NODE
LOOP
RESERVE FAM.OF.GRP(*) AS (GRPS + N.NODE + 25)
FOR I = 1 TO N.NODE, DO
  IF FMLYS LT FAMILY(I)
     LET FMLYS = FAMILY(I)
  REGARDLESS
''
''   SET PROGRAM FAM NUM
''
  LET FAMILY(I) = N.NODE + GRPS + FAMILY(I)
```

```
   LET FAM.OF.GRP(GROUP(I)) = FAMILY(I)
LOOP
LET NGFS = N.NODE + GRPS + FMLYS
IF SPECIFY.OUTPUT LE 1
   FOR I = 1 TO N.NODE, DO
      PRINT 1 LINE WITH I, TRANSMIT.PERCENT(I), RECEIVE.PERCENT(I),
        (GROUP(I) - N.NODE), GROUP(I), (FAMILY(I) - N.NODE - GRPS)
        AND FAMILY(I) AS FOLLOWS
**          **.***        **.***      **(**)     **(**)
   LOOP
   SKIP 1 OUTPUT LINE
REGARDLESS
''
''    RECORD THE NETWORK TOPOLOGY BY READING THE LINK CONNECTIVITIES INTO
''      A 2-DIMENSIONAL INTEGER ARRAY CALLED "LINKABLE".  KEEP TRACK OF
''      HOW MANY LINKS THERE ARE.
''
RESERVE LINKABLE(*,*) AS N.NODE BY N.NODE
LET LINKS = 0
FOR I = 1 TO N.NODE, DO
   FOR J = 1 TO N.NODE, DO
      READ LINKABLE(I,J)
      IF LINKABLE(I,J) GT 0
         LET LINKS = LINKS + 1
      ALWAYS
   LOOP
LOOP
LET LINKS = INT.F(REAL.F(LINKS) / 2.0)
LET LINK.NODE.RATIO = REAL.F(LINKS) / REAL.F(N.NODE)
''
RESERVE LI.NK.NR(*,*) AS N.NODE BY N.NODE
LET LINK.NR = 1
'MORE'
READ FM
READ TO
LET LI.NK.NR(FM,TO) = LINK.NR
LET LI.NK.NR(TO,FM) = LINK.NR
LET LINK.NR = LINK.NR + 1
IF LINK.NR EQ 31
   GO TO LABEL
ALWAYS
GO TO MORE
'LABEL'
''
LET MAX.LINKS.PER.NODE = 0
RESERVE NODE.COUNT(*,*) AS 6 BY N.NODE
LET A = 1
LET B = 1
LET C = 1
LET D = 1
LET E = 1
LET F = 1
FOR I = 1 TO N.NODE, DO
   LET COUNT = 0
   FOR J = 1 TO N.NODE, DO
      IF LINKABLE(I,J) EQ 1
         LET COUNT = COUNT + 1
      ALWAYS
   LOOP
   IF COUNT EQ 1
      LET NODE.COUNT(COUNT,A) = I
      LET A = A + 1
      GO TO OUT
   ALWAYS
   IF COUNT EQ 2
      LET NODE.COUNT(COUNT,B) = I
      LET B = B + 1
      GO TO OUT
   ALWAYS
   IF COUNT EQ 3
      LET NODE.COUNT(COUNT,C) = I
      LET C = C + 1
```

```
      GO TO CUT
   ALWAYS
   IF COUNT EQ 4
      LET NODE.COUNT(COUNT,D) = I
      LET D = D + 1
      GO TO OUT
   ALWAYS
   IF COUNT EQ 5
      LET NODE.COUNT(COUNT,E) = I
      LET E = E + 1
      GO TO OUT
   ALWAYS
   IF COUNT EQ 6
      LET NODE.COUNT(COUNT,F) = I
      LET F = F + 1
      GO TO OUT
   ALWAYS
''
'OUT'
   IF COUNT GT MAX.LINKS.PER.NODE
      LET MAX.LINKS.PER.NODE = COUNT
   ALWAYS
   LET LINKABLE(I,I) = COUNT
LOOP
''
''    READ IN THE LINK ATTENUATIONS AND STORE THESE VALUES IN THE 2-DIM-
''       ENSIONAL REAL ARRAY CALLED "ATTENUATION".
''
RESERVE ATTENUATION(*,*) AS N.NODE BY N.NODE
FOR I = 1 TO N.NODE, DO
   FOR J = 1 TO N.NODE, DO
      IF I NE J AND LINKABLE(I,J) EQ 1
         READ ATTENUATION(I,J)
      ALWAYS
   LOOP
LOOP
''
''    WE CAN NOW OPERATE ON THE ATTENUATIONS JUST READ IN TO PRODUCE THE
''       "ENERGY" ARRAY, THE ENTRIES OF WHICH WILL BE A REPRESENTATION OF
''       THE ENERGY PER BIT REQUIRED TO TRANSMIT A BIT OF DATA OVER A PAR-
''       TICULAR LINK WITH A GIVEN ATTENUATION.  THE "NERGY" ARRAY IS A
''       COPY OF THE ENERGY ARRAY THAT WILL BE DESTRUCTIVELY MANIPULATED
''       WHEN WE CALCULATE THE LINK WEIGHTS BELOW.
''
RESERVE ENERGY(*,*) AS N.NODE BY N.NODE
RESERVE NERGY(*,*) AS N.NODE BY N.NODE
FOR I = 1 TO N.NODE, DO
   FOR J = 1 TO N.NODE, DO
      IF I NE J AND LINKABLE(I,J) EQ 1
         LET ADJUSTED.ATT = ATTENUATION(I,J) - 81.0
         LET ADJUSTED.ATT = ADJUSTED.ATT / 10.0
         LET EN.ERGY = 10.0 ** ADJUSTED.ATT
         LET ENERGY(I,J) = EN.ERGY
         LET NERGY(I,J) = EN.ERGY
      ALWAYS
   LOOP
LOOP
''
''    SINCE THE LINK ATTENUATIONS (AND THEREFORE THE REQUIRED ENERGY PER
''       BIT) REMAIN THE SAME FOR ALL RUNS OF THE SIMULATION WE CAN NOW
''       "SCALE" OR "WEIGHT" THE LINKS.  THESE "LINK.WEIGHTS" ARE ASSIGNED
''       WEIGHTS FROM 1.0 TO 128.0 ACCORDING TO A GEOMETRIC DISTRIBUTION.
''
RESERVE LINK.WEIGHT(*,*) AS N.NODE BY N.NODE
LET WT = 1000000.0
LET SUM = 0
'SEARCH'
FOR I = 1 TO N.NODE, DO
   FOR J = 1 TO N.NODE, DO
      IF I NE J AND NERGY(I,J) LE WT AND LINKABLE(I,J) EQ 1 AND
         NERGY(I,J) NE 0.0
         LET WT = NERGY(I,J)
```

```
        LET IINDEX = I
        LET JINDEX = J
     ALWAYS
  LOOP
LOOP
LET SUM = SUM + 1
IF SUM LE LINKS
  READ LINK.WEIGHT(IINDEX,JINDEX)
  LET LINK.WEIGHT(JINDEX,IINDEX) = LINK.WEIGHT(IINDEX,JINDEX)
  LET NERGY(IINDEX,JINDEX) = 0.0
  LET NERGY(JINDEX,IINDEX) = 0.0
  LET WT = 1000000.0
  GO TO SEARCH
ALWAYS
''
''    READ THE REMAINING INPUT PARAMETERS
''
READ TEST.DURATION
READ MAX.CKTS.IN.SIM
READ SLOTS
READ STARTING.MAX.SLOT.DEPTH
READ ENDING.MAX.SLOT.DEPTH
LET MAX.SLOT.DEPTH = STARTING.MAX.SLOT.DEPTH
READ SLOT.DURATION
READ PROCESSING.TIME
READ PROP.DELAY.TIME
READ MEAN.CKT.ESTAB
LET NODAL.MEAN.CKT.ESTAB = MEAN.CKT.ESTAB
LET MEAN.CKT.ESTAB = MEAN.CKT.ESTAB / REAL.F(N.NODE)
READ MEAN.DURATION.OF.CKT
READ UP.DATE.PERIOD
IF ROUTING.ALGORITHM.SELECTOR EQ 2 AND UP.DATE.PERIOD LE TEST.DURATION
  LET UP.DATE.PERIOD = TEST.DURATION + 1.0
ALWAYS
READ RE.PORT.PERIOD
LET STARTER = 1
''
''    IN.GROUP MEANS THE PERCENTAGE OF GENERATED CIRCUIT REQUIREMENTS
''      THAT WILL NOT LEAVE ITS BASIC GROUP; SIMILARLY FOR IN.FAMILY.
''      NOTE: IF ALL NODES ARE SPECIFIED TO BE MEMBERS OF THE SAME GROUP
''      AND FAMILY THEN VALUES OF IN.GROUP AND IN.FAMILY ARE IGNORED BY
''      THE PROGRAM.
''
READ IN.GROUP
READ IN.FAMILY
''
''    BRK.X.POINT AND BRK.Y.POINT LOCATE THE "KNEE" OF THE CURVE USED
''      TO CALCULATE THE NODE WEIGHT WHICH IS USED IN THE DYNAMIC
''      ROUTING ROUTE CALCULATION.
''
READ BRK.X.POINT
READ BRK.Y.POINT
READ NODE.MAX.SCALE.WEIGHT
RESERVE BEST.PATH(*,*) AS N.NODE BY N.NODE
FOR I = 1 TO N.NODE, DO
  FOR J = 1 TO N.NODE, DO
     READ BEST.PATH(I,J)
  LOOP
LOOP
''
''    PRINT ALL INPUT DATA AS THE OUTPUT HEADER.  THIS IS DONE BY THE
''      "ECHO.PRINT.INPUT.DATA" ROUTINE.
''
IF SPECIFY.OUTPUT LE 1
''
  PERFORM ECHO.PRINT.INPUT.DATA
''
REGARDLESS
''
RELEASE NODE.COUNT(*,*)
RELEASE NERGY(*,*)
''
```

```
RETURN
END ''OF HOUSEKEEPING
''
''   THIS ROUTINE IS CALLED ONLY BY THE HOUSEKEEPING ROUTINE AND THEN
''      ONLY WHEN WE DESIRE AN ECHO PRINT OF SOME OF THE INPUT DATA.
''
ROUTINE FOR ECHO.PRINT.INPUT.DATA
''
   SKIP 1 OUTPUT LINE
   IF ROUTING.ALGORITHM.SELECTOR EQ 1
      PRINT 3 LINES AS FOLLOWS
-------------------------------------------------------------------------------
-            THIS SIMULATION IS FOR DYNAMIC BEST PATH ROUTING                 -
-------------------------------------------------------------------------------
      SKIP 1 OUTPUT LINE
   ALWAYS
   IF ROUTING.ALGORITHM.SELECTOR EQ 2
      PRINT 3 LINES AS FOLLOWS
-------------------------------------------------------------------------------
-       THIS SIMULATION IS FOR STATIC BEST PATH LEAST HOP ROUTING             -
-------------------------------------------------------------------------------
      SKIP 1 OUTPUT LINE
   ALWAYS
   PRINT 1 LINE WITH N.NODE AS FOLLOWS
THE NUMBER OF NODES IN THE NETWORK IS **
   SKIP 1 OUTPUT LINE
''
   PRINT 1 LINE WITH LINKS AS FOLLOWS
THE NUMBER OF LINKS IN THE NETWORK IS **
   SKIP 1 OUTPUT LINE
''
   PRINT 1 LINE WITH LINK.NODE.RATIO AS FOLLOWS
THE RATIO OF LINKS TO NODES FOR THE NETWORK IS **.****
   SKIP 1 OUTPUT LINE
''
   PRINT 2 LINES WITH TEST.DURATION AND MAX.CKTS.IN.SIM AS FOLLOWS
THE SIMULATION WILL RUN FOR A SIMULATION TIME OF ****.** SECONDS,
   OR UNTIL SUCH TIME AS WE HAVE ATTEMPTED TO ESTABLISH ***** CIRCUITS.
   SKIP 1 OUTPUT LINE
''
   PRINT 1 LINE WITH SLOTS AS FOLLOWS
THE NUMBER OF TIME SLOTS PER FRAME = **
   SKIP 1 OUTPUT LINE
''
   PRINT 5 LINES WITH STARTING.MAX.SLOT.DEPTH AND ENDING.MAX.SLOT.DEPTH
      AS FOLLOWS
TIME SLOTS USED TO RECEIVE MAY BE ALLOWED TO RECEIVE BETWEEN ** AND **
   SIGNALS SIMULTANEOUSLY.  THE ACTUAL DEPTH OF THE "USE" ARRAY FOR
   EACH NODE IS ALWAYS ONE LEVEL GREATER THAN THE ASSIGNED MAX.SLOT.DEPTH
   BECAUSE OF THE REQUIREMENT TO ALWAYS BE ABLE TO RECEIVE POSSIBLE
   INTERNODAL SERVICE MESSAGES IN NON-TRANSMIT SLOTS.
   SKIP 1 OUTPUT LINE
''
   PRINT 1 LINE WITH MAX.LINKS.PER.NODE AND MAX.LINKS.PER.NODE AS FOLLOWS
THERE IS AT LEAST ONE NODE MAINTAINING * LINKS WITH * OTHER NODES.
   SKIP 1 OUTPUT LINE
   FOR I = 1 TO 6, DO
      IF NODE.COUNT(I,1) NE 0
         PRINT 2 LINES WITH I AND I AS FOLLOWS
THE FOLLOWING NODE(S) CLAIM(S) * NEIGHBORS (I.E. MAINTAINS * LINKS):
       NODE(S)
         FOR J = 1 TO N.NODE, DO
            IF NODE.COUNT(I,J) EQ 0
               SKIP 1 OUTPUT LINE
               GO TO RESUME
            ALWAYS
            PRINT 1 LINE WITH NODE.COUNT(I,J) AS FOLLOWS
           **
         LOOP
         SKIP 1 OUTPUT LINE
      ALWAYS
'RESUME'
```

```
    LOOP
    SKIP 1 OUTPUT LINE
''
    PRINT 5 LINES AS FOLLOWS
THE CONTENTS OF THE ATTENUATION ARRAY ARE:
    +
     +TO          1           2           3           4           5           6           7
FROM+
-----+--------------------------------------------------------------------------------------
    FOR I = 1 TO N.NODE, DO
       PRINT 1 LINE WITH I, ATTENUATION(I,1), ATTENUATION(I,2),
          ATTENUATION(I,3), ATTENUATION(I,4), ATTENUATION(I,5),
          ATTENUATION(I,6) AND ATTENUATION(I,7) AS FOLLOWS
 **   + *****.** *****.** *****.** *****.** *****.** *****.** *****.**
    LOOP
    SKIP 1 OUTPUT LINE
    PRINT 5 LINES AS FOLLOWS
    ATTENUATION ARRAY (CONT.):
    +
     +TO          8           9          10          11          12          13
FROM+
-----+--------------------------------------------------------------------------------------
    FOR I = 1 TO N.NODE, DO
       PRINT 1 LINE WITH I, ATTENUATION(I,8), ATTENUATION(I,9),
          ATTENUATION(I,10), ATTENUATION(I,11), ATTENUATION(I,12) AND
          ATTENUATION(I,13) AS FOLLOWS
 **   + *****.** *****.** *****.** *****.** *****.** *****.**
    LOOP
    SKIP 2 OUTPUT LINES
''
    PRINT 5 LINES AS FOLLOWS
THE CONTENTS OF THE ENERGY ARRAY ARE:
    +
     +TO          1           2           3           4           5           6           7
FROM+
-----+--------------------------------------------------------------------------------------
    FOR I = 1 TO N.NODE, DO
       PRINT 1 LINE WITH I, ENERGY(I,1), ENERGY(I,2), ENERGY(I,3),
          ENERGY(I,4), ENERGY(I,5), ENERGY(I,6) AND ENERGY(I,7) AS FOLLOWS
 **   + ******.* ******.* ******.* ******.* ******.* ******.* ******.*
    LOOP
    SKIP 1 OUTPUT LINE
    PRINT 5 LINES AS FOLLOWS
ENERGY ARRAY (CONT.):
    +
     +TO          8           9          10          11          12          13
FROM+
-----+--------------------------------------------------------------------------------------
    FOR I = 1 TO N.NODE, DO
       PRINT 1 LINE WITH I, ENERGY(I,8), ENERGY(I,9), ENERGY(I,10),
          ENERGY(I,11), ENERGY(I,12) AND ENERGY(I,13) AS FOLLOWS
 **   + ******.* ******.* ******.* ******.* ******.* ******.*
    LOOP
    SKIP 2 OUTPUT LINES
''
    PRINT 5 LINES AS FOLLOWS
THE CONTENTS OF THE LINK.WEIGHT ARRAY ARE:
    +
     +TO          1           2           3           4           5           6           7
FROM+
-----+--------------------------------------------------------------------------------------
    FOR I = 1 TO N.NODE, DO
       PRINT 1 LINE WITH I, LINK.WEIGHT(I,1), LINK.WEIGHT(I,2),
          LINK.WEIGHT(I,3), LINK.WEIGHT(I,4), LINK.WEIGHT(I,5),
          LINK.WEIGHT(I,6) AND LINK.WEIGHT(I,7) AS FOLLOWS
 **   + ******.* ******.* ******.* ******.* ******.* ******.* ******.*
    LOOP
    SKIP 1 OUTPUT LINE
    PRINT 5 LINES AS FOLLOWS
LINK.WEIGHT ARRAY (CONT.):
    +
     +TO          8           9          10          11          12          13
```

```
FROM+
-----+------------------------------------------------------------------
   FOR I = 1 TO N.NODE, DO
     PRINT 1 LINE WITH I, LINK.WEIGHT(I,8), LINK.WEIGHT(I,9),
       LINK.WEIGHT(I,10), LINK.WEIGHT(I,11), LINK.WEIGHT(I,12) AND
       LINK.WEIGHT(I,13) AS FOLLOWS
 **  + ******.* ******.* ******.* ******.* ******.* ******.*
   LOOP
   SKIP 2 OUTPUT LINES
   IF ROUTING.ALGORITHM.SELECTOR EQ 2
     PRINT 6 LINES AS FOLLOWS
THE CONTENTS OF THE STATIC BEST PATH MATRIX USED THROUGHOUT THIS LEAST
   HOP SIMULATION ARE:
   +
  +TO    1     2     3     4     5     6     7     8     9    10    11    12    13
FROM+
-----+------------------------------------------------------------------
     FOR I = 1 TO N.NODE, DO
       PRINT 1 LINE WITH I, BEST.PATH(I,1), BEST.PATH(I,2),
         BEST.PATH(I,3), BEST.PATH(I,4), BEST.PATH(I,5), BEST.PATH(I,6),
         BEST.PATH(I,7), BEST.PATH(I,8), BEST.PATH(I,9), BEST.PATH(I,10),
         BEST.PATH(I,11), BEST.PATH(I,12) AND BEST.PATH(I,13) AS FOLLOWS
 **  +   **    **    **    **    **    **    **    **    **    **    **    **    **
     LOOP
     SKIP 2 OUTPUT LINES
   ALWAYS
''
   PRINT 15 LINES WITH SLOT.DURATION, PROCESSING.TIME, PROP.DELAY.TIME,
     MEAN.CKT.ESTAB , NODAL.MEAN.CKT.ESTAB, MEAN.DURATION.OF.CKT AND
     UP.DATE.PERIOD AND RE.PORT.PERIOD AS FOLLOWS
TIMING PARAMETERS (NOTE: UNITS OF TIME IN PROGRAM AND BELOW ARE SECONDS)
   THE DURATION OF A TIME SLOT IS *.******
   THE PROCESSING TIME FOR A VOICE PACKET OR SERVICE MESSAGE IS *.******
   THE PROPAGATION DELAY TIME BETWEEN ANY TWO NODES IS *.******
   NEW REQUIREMENTS FOR VOICE CIRCUITS ARE GENERATED OVER THE NETWORK AS
     A WHOLE WITH AN EXPONENTIAL DISTRIBUTION FUNCTION HAVING A MEAN TIME
     BETWEEN CIRCUIT REQUIREMENTS OF ***.****** SECONDS, AND THE MEAN
     TIME BETWEEN THE ORIGINATION OF THE NEW CIRCUIT REQUIREMENTS FOR
     EACH NODE IN THE NETWORK IS ***.****** SECONDS.
   ONCE ESTABLISHED, TWO-WAY VIRTUAL VOICE CIRCUITS REMAIN IN EFFECT AS
     DETERMINED BY AN EXPONENTIAL DISTRIBUTION FUNCTION HAVING A MEAN
     CALL DURATION OF ***.****** SECONDS.
   IF WE ARE ROUTING DYNAMICALLY, THEN THE DIJKSTRA ROUTING EVENT UPDATES
     BEST PATH ROUTE INFORMATION EVERY ***.****** SECONDS.
   THE PROGRAM PRINTS INTERMEDIATE RESULTS EVERY ***.****** SECONDS.
   SKIP 1 OUTPUT LINE
''
   PRINT 4 LINES WITH IN.GROUP AND IN.FAMILY AS FOLLOWS
AT LEAST **.*% OF CIRCUIT REQUIREMENTS ARE BETWEEN NODES IN THE SAME
   BASIC GROUP.
AT LEAST **.*% OF CIRCUIT REQUIREMENTS ARE BETWEEN NODES IN THE SAME
   FAMILY.
   SKIP 1 OUTPUT LINE
''
   PRINT 4 LINES WITH NODE.MAX.SCALE.WEIGHT, BRK.X.POINT AND BRK.Y.POINT
     AS FOLLOWS
WHEN WE ARE SIMULATING DYNAMIC ROUTING,
   THE MAXIMUM NODE SCALE WEIGHT = ******.*
   THE X-COORDINATE OF THE NODE WEIGHT BREAK POINT IS BIN NR. ****
   THE Y-COORDINATE OF THE NODE WEIGHT BREAK POINT IS:         ****
   SKIP 1 OUTPUT LINE
''
RETURN
END ''OF ECHC.PRINT.INPUT.DATA
''
''    THIS ROUTINE RESERVES AND SETS UP THE ARRAYS ASSOCIATED WITH THE
''      DYNAMIC ROUTING PORTION OF THE PROGRAM.
''
ROUTINE FOR ARRAY.INITIALIZATION
''
''    THE DIJKSTRA ARRAY HOLDS A REAL NON-NEGATIVE NUMBER INDICATING THE
''      TOTAL OVERALL LINK "DISTANCE" FROM EACH NODE TO EVERY OTHER NODE
```

```
''      IN THE NETWORK.  INITIALLY, IF A DIRECT LINK EXISTS BETWEEN TWO
''      NODES WE SHALL ASSIGN A VALUE OF 1.0 AND IF A DIRECT LINK DOES
''      NOT EXIST, WE SHALL ASSIGN A VALUE OF 999999.9.  THE VALUES IN
''      THIS ARRAY WILL CHANGE DURING THE SIMULATION AS INDIVIDUAL LINK
''      WEIGHTS CHANGE TO REFLECT VARYING DEGREES OF LINK, NODE AND NET-
''      WORK LOADING.
''
DEFINE SLOPE1 AND SLOPE2 AS REAL VARIABLES
''
RESERVE DIJKSTRA(*,*) AS N.NODE BY N.NODE
FOR I = 1 TO N.NODE, DO
   FOR J = 1 TO N.NODE, DO
      IF I EQ J
         LET DIJKSTRA(I,J) = 0.0
      ALWAYS
      IF I NE J AND LINKABLE(I,J) EQ 0
         LET DIJKSTRA(I,J) = 999999.9
      ALWAYS
      IF I NE J AND LINKABLE(I,J) EQ 1
         LET DIJKSTRA(I,J) = 1.0
      ALWAYS
   LOOP
LOOP
''
''   THE DISTANCE ARRAY HOLDS A REAL NON-NEGATIVE NUMBER REPRESENTING
''      THE "DISTANCE" OVER ONE LINK FROM ONE NODE TO ONE OF ITS NEIGH-
''      BORING NODES.  INITIALLY, ALL LINK WEIGHTS ARE SET TO 1.0 ON THE
''      DIRECT LINKS AND TO A LARGE, POSITIVE REAL NUMBER WHEN NO DIRECT
''      LINK EXISTS.
''
RESERVE DISTANCE(*,*) AS N.NODE BY N.NODE
FOR I = 1 TO N.NODE, DO
   FOR J = 1 TO N.NODE, DO
      LET DISTANCE(I,J) = DIJKSTRA(I,J)
   LOOP
LOOP
''
''   THE BEST.PATH ARRAY HOLDS AN INTEGER NODE IDENTIFICATION NUMBER OF
''      THE BEST PATH NEIGHBOR FROM ANY GIVEN NODE TO ANY OTHER NODE IN
''      THE NETWORK.  UNTIL SUCH TIME AS THE "DIJK.MANIPULATION" EVENT IS
''      CALLED , WE CAN ONLY ASSIGN THE DIRECT LINKS AS SINGLE HOP BEST
''      PATHS.
''
RESERVE BEST.PATH(*,*) AS N.NODE BY N.NODE
FOR I = 1 TO N.NODE, DO
   FOR J = 1 TO N.NODE, DO
      IF LINKABLE(I,J) EQ 1
         LET BEST.PATH(I,J) = J
      ALWAYS
   LOOP
LOOP
''
''   THE NODE.SCALE ARRAY HOLDS THE SCALED VALUE OF THE NODE WEIGHT
''      SCALED INTO "BINS" NUMBERED FROM 1 TO 128.  THESE SCALED WEIGHTS
''      ARE USED IN THE CALCULATION OF THE LINK DISTANCES IN THE COM-
''      PUTE.CURRENT.DISTANCES ROUTINE.
''
RESERVE NODE.SCALE(*) AS 128
IF BRK.X.POINT EQ 0 OR BRK.X.POINT EQ 1
   LET BRK.X.POINT = 0
   GO TO ASSIGN.VALUES
ALWAYS
FOR I = 1 TO BRK.X.POINT, DO
   LET SLOPE1 = REAL.F(BRK.Y.POINT) / REAL.F(BRK.X.POINT)
   LET NODE.SCALE(I) = SLOPE1 * REAL.F(I)
   IF NODE.SCALE(I) EQ 0.0
      LET NODE.SCALE(I) = 0.0
   ALWAYS
LOOP
'ASSIGN.VALUES'
IF BRK.X.POINT LE 127
   LET SLOPE2 = (NODE.MAX.SCALE.WEIGHT - REAL.F(BRK.Y.POINT)) / (128.0 -
```

```
     REAL.F(BRK.X.POINT))
   FOR I = (BRK.X.POINT + 1) TO 128, DO
     LET NODE.SCALE(I) = (SLOPE2 * REAL.F(I - BRK.X.POINT)) +
       REAL.F(BRK.Y.POINT)
   LOOP
ALWAYS
''
''   PRINT THESE ARRAYS TO ENSURE THEY WERE SET UP PROPERLY.
''
IF SPECIFY.OUTPUT EQ 0 AND PRT LT 3
   PRINT 1 LINE WITH TIME.V AS FOLLOWS
ARRAY.INITIALIZATION ROUTINE CALLED AT TIME.V = ****.******
   SKIP 1 OUTPUT LINE
   PRINT 5 LINES AS FOLLOWS
THE CONTENTS OF THE DIJKSTRA MATRIX ARE:
   +
    +TO          1          2          3          4          5          6          7
FROM+
-----+----------------------------------------------------------------------------
   FOR I = 1 TO N.NODE, DO
     PRINT 1 LINE WITH I, DIJKSTRA(I,1), DIJKSTRA(I,2), DIJKSTRA(I,3),
       DIJKSTRA(I,4), DIJKSTRA(I,5), DIJKSTRA(I,6) AND DIJKSTRA(I,7)
       AS FOLLOWS
 **   + ******.* ******.* ******.* ******.* ******.* ******.* ******.*
   LOOP
   SKIP 1 OUTPUT LINE
   PRINT 5 LINES AS FOLLOWS
   CONTENTS OF THE DIJKSTRA MATRIX (CONT.):
   +
    +TO          8          9         10         11         12         13
FROM+
-----+----------------------------------------------------------------------------
   FOR I = 1 TO N.NODE, DO
     PRINT 1 LINE WITH I, DIJKSTRA(I,8), DIJKSTRA(I,9), DIJKSTRA(I,10),
       DIJKSTRA(I,11), DIJKSTRA(I,12) AND DIJKSTRA(I,13) AS FOLLOWS
 **   + ******.* ******.* ******.* ******.* ******.* ******.*
   LOOP
   SKIP 2 OUTPUT LINES
''
   PRINT 5 LINES AS FOLLOWS
THE CONTENTS OF THE DISTANCE MATRIX ARE:
   +
    +TO          1          2          3          4          5          6          7
FROM+
-----+----------------------------------------------------------------------------
   FOR I = 1 TO N.NODE, DO
   PRINT 1 LINE WITH I, DISTANCE(I,1), DISTANCE(I,2), DISTANCE(I,3),
     DISTANCE(I,4), DISTANCE(I,5), DISTANCE(I,6) AND DISTANCE(I,7)
     AS FOLLOWS
 **   + ******.* ******.* ******.* ******.* ******.* ******.* ******.*
   LOOP
   SKIP 1 OUTPUT LINE
   PRINT 5 LINES AS FOLLOWS
   CONTENTS OF THE DISTANCE MATRIX (CONT.):
   +
    +TO          8          9         10         11         12         13
FROM+
-----+----------------------------------------------------------------------------
   FOR I = 1 TO N.NODE, DO
     PRINT 1 LINE WITH I, DISTANCE(I,8), DISTANCE(I,9), DISTANCE(I,10),
       DISTANCE(I,11), DISTANCE(I,12) AND DISTANCE(I,13) AS FOLLOWS
 **   + ******.* ******.* ******.* ******.* ******.* ******.*
   LOOP
   SKIP 2 OUTPUT LINES
''
   PRINT 5 LINES AS FOLLOWS
THE CONTENTS OF THE BEST.PATH MATRIX ARE:
   +
    +TO    1     2     3     4     5     6     7     8     9    10    11    12    13
FROM+
-----+----------------------------------------------------------------------------
   FOR I = 1 TO N.NODE, DO
```

```
     PRINT 1 LINE WITH I, BEST.PATH(I,1), BEST.PATH(I,2), BEST.PATH(I,3),
       BEST.PATH(I,4), BEST.PATH(I,5), BEST.PATH(I,6), BEST.PATH(I,7),
       BEST.PATH(I,8), BEST.PATH(I,9), BEST.PATH(I,10), BEST.PATH(I,11),
       BEST.PATH(I,12) AND BEST.PATH(I,13) AS FOLLOWS
 **  +  **   **   **   **   **   **   **   **   **   **   **   **   **
   LOOP
   SKIP 2 OUTPUT LINES
REGARDLESS
''
IF SPECIFY.OUTPUT EQ 0 AND PRT LE 3 AND MAX.SLOT.DEPTH EQ
   STARTING.MAX.SLOT.DEPTH
   PRINT 2 LINES AS FOLLOWS
THE CONTENTS OF THE NODE.SCALE ARRAY ARE:
    CALCULATED VALUE (BIN NR.)            SCALED VALUE (BIN CONTENTS)
   FOR I = 1 TO 128, DO
     PRINT 1 LINE WITH I AND NODE.SCALE(I) AS FOLLOWS
              ****                                  ****.**
   LOOP
   SKIP 2 OUTPUT LINES
''
REGARDLESS
''
RETURN
END ''OF ARRAY.INITIALIZATION
''
''    THIS ROUTINE HALTS THE PROGRAM AND PRINTS IMPORTANT STATISTICS
''      AT PERIODIC INTERVALS THROUGHOUT THE SIMULATION.
''
EVENT STOP.SIMULATION
''
DEFINE ACT.CAP AS A REAL VARIABLE
LET REPORT.COUNTER = REPORT.COUNTER + 1
''
IF TIME.V GE TEST.DURATION
   LET PRNT = 1
ALWAYS
''
IF REPORT.COUNTER EQ 1
   PRINT 1 DOUBLE LINE AS FOLLOWS
REPORT   TIME.V   ACT CKTS   AVG CKTS   AVG NR    AVG       MEAN    VARIANCE
D.DEV     MAX       MIN
   PRINT 1 DOUBLE LINE AS FOLLOWS
NUMBER   EQUALS    ACTIVE     ACTIVE     HOPS    ENERGY   ACTIVE    ACTIVE
TIVE    ACTIVE   ACTIVE
   PRINT 1 DOUBLE LINE AS FOLLOWS
------   ------   --------   --------   ------   ------   ------   --------
-----   ------   -------
   SKIP 1 OUTPUT LINE
   LET AVG.ACTIVE = REAL.F(ACTIVE)
   GO TO LEAVE.THIS.CALCULATION
ALWAYS
LET AVG.ACTIVE = (REAL.F(ACTIVE) + AVG.ACTIVE) / 2.0
'LEAVE.THIS.CALCULATION'
LET FRACT.OF.SUCCESSFUL.CALLS = (REAL.F(CKT.ESTAB) / REAL.F(CKT.TOTAL -
   UP.ROUTE)) * 100.0
''
IF LTD.PRINT EQ 1
   PRINT 1 DOUBLE LINE WITH REPORT.COUNTER, TIME.V, ACTIVE, AVG.ACTIVE,
     HOP.AVG, (E.SUB.K.BAR / 100000.0), CUM.MEAN, CUM.VARIANCE,
     CUM.STD.DEVIATION, MAX.ACTIVE AND MIN.ACTIVE AS FOLLOWS
 ***      ****.*     ***        ***.**     **.**     **.**     ***.**     ***.**
*.**     ***.*     ***.*
   SKIP 1 OUTPUT LINE
   IF TIME.V GE TEST.DURATION
     GO TO FULL.REPORT
   ALWAYS
   GO TO DEPARTURE
ALWAYS
''
'FULL.REPORT'
LET PRNT.FLAG = 0
IF SPECIFY.OUTPUT LT 1 AND PRNT LE 5
```

```
   START NEW PAGE
   PRINT 1 LINE WITH REPORT.COUNTER AND TIME.V AS FOLLOWS
XXXXX START INTERMEDIATE REPORT NR. **** FOR TIME.V = ****.****** XXXXX
   SKIP 2 OUTPUT LINES
   PRINT 49 LINES WITH CKT.SUM, CKT.TOTAL, CKT.ESTAB, CKT.DISESTAB,
     CKT.FAILED, TOT.DIJK.CALLED, BACKTRACK.OR.LOOPBACK, ACT.LOOP.REMOVE,
     UP.ROUTE, DOWN.ROUTE, ACTIVE, HOP.AVG,  TOT.HOP.GREATEST,
     HOP.GREATEST, CKT.GREATEST, AVG.TIME.EST, LONG.TIME.EST,
     CKT.LONG.TIME.EST, AVG.DURATION, P.BD.COUNTER, AVG.P.BD, LONG.P.BD,
     C.BD.COUNTER, AVG.C.BD, LONG.C.BD, AVG.BD.TIME, MAX.SLOT.DEPTH,
     AVG.ACTIVE, FRACT.OF.SUCCESSFUL.CALLS,
     (100.0 - FRACT.OF.SUCCESSFUL.CALLS) AND E.SUB.K.BAR AS FOLLOWS
OVERALL CUMULATIVE STATISTICS FOR THIS RUN OF THE SIMULATION:
  TOTAL NUMBER OF CIRCUIT REQUIREMENTS GENERATED = *****                    @
  TOTAL NUMBER OF CIRCUIT REQUIREMENTS GENERATED = ***** ckt Total          #
  TOTAL NUMBER OF CIRCUITS ESTABLISHED = ***** ckt estab                    #
  TOTAL NUMBER OF CIRCUITS THAT WERE ONCE ESTABLISHED AND
    THAT HAVE NOW BEEN DISESTABLISHED = ***** ckt disestab                  #
  TOTAL NUMBER OF CIRCUITS UNABLE TO BE ESTABLISHED = ***** ckt.failed     #
  THE TOTAL NUMBER OF TIMES THE DIJK.MANIPULATION EVENT HAS BEEN CALLED
    TO UPDATE THE BEST PATH ROUTES = ****** Tot                            @
  THE TOTAL NUMBER OF CIRCUITS THAT WERE FOUND TO BACKTRACK OR TO LOOP
    BACK ACROSS THEMSELVES AS A RESULT OF UPDATED BEST PATHS = *** back    #
  AT THIS INSTANT THERE ARE:
up  ** VIRTUAL CIRCUITS IN THE PROCESS OF REMOVING LOOPS
[illegible]** VIRTUAL CIRCUITS IN THE PROCESS OF BEING ESTABLISHED
down** VIRTUAL CIRCUITS IN THE PROCESS OF BEING DISESTABLISHED
act ** VIRTUAL CIRCUITS ESTABLISHED AND ACTIVELY CARRYING VOICE TRAFFIC
CUMULATIVE STATISTICS CONCERNING ESTABLISHED CIRCUITS:
  AVERAGE NUMBER OF HOPS PER ESTABLISHED CIRCUIT = **.*** Act up           #
  ** CIRCUIT(S) TOOK THE LARGEST NUMBER OF HOPS, I.E. **.* HOPS TO         #
    ESTABLISH THE VIRTUAL CIRCUIT.  CIRCUIT NR. *****, IS THE MOST RECENT
    CIRCUIT TO USE THE LARGEST NUMBER OF HOPS. down                        #
  AVERAGE TIME TO ESTABLISH A CIRCUIT = **.****** SECONDS                  #
  LONGEST TIME TO ESTABLISH A CIRCUIT = **.****** FOR CKT.NR *****         #
  ACTUAL OBSERVED AVG.DURATION OF AN ESTABLISHED CIRCUIT = ****.****** #
CUMULATIVE STATISTICS CONCERNING DISESTABLISHED CIRCUITS:
  PARTIALLY ESTABLISHED CIRCUITS:
    TOTAL NUMBER OF ONCE PARTIALLY ESTABLISHED CIRCUITS
      THAT HAVE NOW BEEN BROKEN DOWN = *****                               #
    AVERAGE TIME TO BREAK DOWN A PARTIAL CIRCUIT = **.******               #
    LONGEST TIME TO BREAK DOWN A PARTIAL CIRCUIT = **.******               #
  ONCE FULLY ESTABLISHED CIRCUITS:
    TOTAL NUMBER OF ONCE FULLY ESTABLISHED CIRCUITS
      THAT HAVE NOW BEEN BROKEN DOWN = *****                               #
    AVERAGE TIME TO BREAK DOWN A COMPLETED CIRCUIT = **.******             #
    LONGEST TIME TO BREAK DOWN A COMPLETED CIRCUIT = **.******             #
  OVERALL STATS ON ALL CIRCUITS DISESTABLISHED:
    AVERAGE TIME TO BREAK DOWN ALL TYPES OF CIRCUITS = **.******           #
FINALLY RECALL THAT THE MAXIMUM SLOT STACKING DEPTH = **                   @

THE AVERAGE NUMBER OF ACTIVE CALLS IS ***.**    @                          #
THE PERCENTAGE OF CALLS ESTAB WITH RESPECT TO CALLS ATTEMPTED IS ***.*%.
 "      "      "    "   FAILED  "      "     "    "      "     "  ***.*%.

THE AVERAGE TOTAL ENERGY OF EACH CIRCUIT IS APPROX. ********.* JOULES. #

WHERE,  @  ==>  IDENTIFIES VALUES COLLECTED OVER THE ENTIRE SIMULATION.
        #  ==>  IDENTIFIES VALUES COLLECTED AFTER THE COUNTERS WERE
                  CLEARED TO REMOVE THE EFFECTS OF THE START-UP TRANS-
                  IENT BEHAVIOR.
  SKIP 3 OUTPUT LINES
ALWAYS
IF SPECIFY.OUTPUT LT 1 AND PRNT LE 3
  FOR N = 1 TO N.NODE, DO
    IF LINE.V GT 73
      START NEW PAGE
    ALWAYS
''
''   COUNT AND PRINT THE TIME SLOT USE STATISTICS.
''
    LET NIL = 0
```

```
      LET T = 0
      LET R = 0
      LET RS = 0
      RESERVE TSLT(*) AS SLOTS
      FOR S = 1 TO SLOTS, DO
         IF USE(N,S,4) GE 1
            LET R = R + USE(N,S,4)
            LET RS = RS + 1
            LET TSLT(S) = USE(N,S,4)
            GO TO ESCAPE
         ALWAYS
         IF USE(N,S,1) GT 0
            LET T = T + 1
            LET TSLT(S) = 10000 + USE(N,S,3)
            GO TO ESCAPE
         ALWAYS
         IF USE(N,S,1) EQ 0 AND USE(N,S,4) EQ 0
            LET NIL = NIL + 1
            LET TSLT(S) = 0
         ALWAYS
'ESCAPE'
      LOOP
      PRINT 2 LINES WITH N, NIL, T, R AND RS AS FOLLOWS
NODE ** HAS ** EMPTY SLOTS, ** TRANSMIT SLOTS, AND HAS ** RECEIVE SIG-
   NALS STACKED IN ** RECEIVE SLOTS.
      SKIP 2 OUTPUT LINES
''
''      PRINT THE TIME SLOT ASSIGNMENTS AT EACH NODE IF THE PRINTING FLAG
''         IS 1 (AND THE SPECIAL PRINTING VARIABLE IS 0).  NORMALLY THIS IN-
''         FORMATION IS ONLY PRINTED TO ASSIST IN DEBUGGING THE OPERATION OF
''         THE PROGRAM, AND THEN THE SLOT ASSIGNMENTS ARE ONLY PRINTED AFTER
''         THE FIRST AND LAST QUARTERS OF EACH RUN OF THE SIMULATION.
''
      IF PRNT.FLAG LE 1
         PRINT 1 LINE WITH TSLT(1), TSLT(2), TSLT(3), TSLT(4), TSLT(5),
            TSLT(6), TSLT(7), TSLT(8), TSLT(9), TSLT(10), TSLT(11) AND
            TSLT(12) AS FOLLOWS
 ***** ***** ***** ***** ***** ***** ***** ***** ***** ***** ***** *****
         PRINT 2 LINES AS FOLLOWS
 -----------------------------------------------------------------------
    1     2     3     4     5     6     7     8     9    10    11    12
         SKIP 4 OUTPUT LINES
      ALWAYS
      RELEASE TSLT(*)
   LOOP
   SKIP 2 OUTPUT LINES
ALWAYS
''
''      FIND THE TOTAL NUMBER OF TRANSMIT SIGNALS PER SLOT IN THE NETWORK.
''
PRINT 1 LINE AS FOLLOWS
        SUMMARY OF THE NUMBER OF TRANSMIT SIGNALS PER SLOT IN THE NETWORK.
SKIP 1 OUTPUT LINE
LET TOT.XMIT = 0
FOR J = 1 TO SLOTS, DO
   LET XMIT = 0
   FOR N = 1 TO N.NODE, DO
      IF USE(N,J,1) GT 0
         LET XMIT = XMIT + 1
         LET TOT.XMIT = TOT.XMIT + 1
      ALWAYS
   LOOP
   PRINT 1 LINE WITH J AND XMIT AS FOLLOWS
        THE NUMBER OF TRANSMIT SIGNALS IN SLOT ** OF THE NETWORK IS **
LOOP
SKIP 2 OUTPUT LINES
LET ACT.CAP = (REAL.F(TOT.XMIT) / THEO.CAP) * 100.0
PRINT 3 LINES WITH TOT.XMIT, INT.F(THEO.CAP) AND ACT.CAP AS FOLLOWS
COUNTERS SHOW THAT THERE ARE PRESENTLY *** OF A POSSIBLE *** TOTAL NUM-
   BER OF TRANSMIT SIGNALS IN THE NETWORK.  THEREFORE THE NETWORK IS
   OPERATING AT APPROXIMATELY ***.** PERCENT OF ITS MAXIMUM CAPACITY.
SKIP 2 OUTPUT LINES
```

```
IF TIME.V GE TEST.DURATION
   START NEW PAGE
   PRINT 9 LINES AS FOLLOWS
LINK USAGE STATISTICS AT END OF SIMULATION.  THESE FIGURES REPRESENT THE
NUMBER OF TIMES EACH LINK CARRIED AN ESTABLISHED CIRCUIT AFTER THE COUN-
TERS WERE RESET TO REMOVE THE TRANSIENT BEHAVIOR OBSERVED DURING THE
SIMULATION START-UP. THEREFORE THESE FIGURES REPRESENT THE STEADY-STATE.

BIDIRECTIONAL                              NUMBER
    LINK                LINK                 OF
FM/TO   TO/FM   ATTENUATION   TIMES USED
-------------   -----------   ----------
   SKIP 1 OUTPUT LINE
   LET COUNTER = 1
'PRINT.NEXT.LINE'
   LET MAX.VAL = -1
   FOR I = 1 TO LINKS, DO
      IF LIN.K.USED(I) GE MAX.VAL
         LET HOLDER = I
      ALWAYS
   LOOP
   FOR I = 1 TO N.NODE, DO
      FOR J = 1 TO N.NODE, DO
         IF LINKABLE(I,J) EQ 1 AND LI.NK.NR(I,J) EQ HOLDER
            PRINT 1 LINE WITH I, J, J, I, ATTENUATION(I,J) AND
               LIN.K.USED(HOLDER) AS FOLLOWS
**/** - **/**      ****.**       *****
            SKIP 1 OUTPUT LINE
            GO TO FLYING.LEAP
         ALWAYS
      LOOP
   LOOP
'FLYING.LEAP'
   LET LIN.K.USED(HOLDER) = -2
   LET COUNTER = COUNTER + 1
   IF COUNTER EQ LINKS - 1
      GO TO FLEE
   ALWAYS
   GO TO PRINT.NEXT.LINE
ALWAYS
''
'FLEE'
PRINT 1 LINE WITH REPORT.COUNTER AS FOLLOWS
XXXXXXXXXXXXXXXXXX END INTERMEDIATE REPORT NR. ****  XXXXXXXXXXXXXXXXXXX
''
''    TEST TO SEE IF ALL INTERMEDIATE REPORTS FOR THIS ITERATION OF THE
''       SIMULATION HAVE BEEN MADE. IF SO, WE CAN CALL THE "DESTRUCTION"
''       ROUTINE AND SEND EXECUTION BACK TO THE MAIN PROGRAM WHERE ANOTHER
''       ITERATION FOR A NEW PARAMETER, OR SET OF PARAMETERS, MAY BE
''       INITIATED.  IF ALL INTERMEDIATE REPORTS HAVE NOT BEEN MADE,
''       THEN WE CAN SCHEDULE THE NEXT "STOP.SIMULATION".
''
'DEPARTURE'
IF (TIME.V + RE.PORT.PERIOD) LE TEST.DURATION
   SCHEDULE A STOP.SIMULATION IN RE.PORT.PERIOD UNITS
ALWAYS
IF (TIME.V + RE.PORT.PERIOD) GT TEST.DURATION
   SCHEDULE A STOP.SIMULATION IN (TIME.V + RE.PORT.PERIOD -
      TEST.DURATION) UNITS
ALWAYS
IF TIME.V LT TEST.DURATION AND PRNT LE 3
   START NEW PAGE
ALWAYS
''
LET PRNT = 5
''
IF TIME.V GE TEST.DURATION
   IF SPECIFY.OUTPUT GE 1
      PERFORM SPECIAL.OUTPUT
   REGARDLESS
   PERFORM DESTRUCTION
REGARDLESS
```

```
''
RETURN
END ''OF STOP.SIMULATION
''
''   THIS EVENT RESETS SOME OF THE COUNTERS TO REMOVE THE EFFECTS OF THE
''     TRANSIENT BEHAVIOR OBSERVED AS THE NETWORK BEGINS ESTABLISHING
''     CIRCUITS.  THUS THE SUCCEEDING PERIODIC REPORTS GIVE A MORE AC-
''     CURATE REPRESENTATION OF THE NETWORK"S STEADY-STATE PERFORMANCE.
''
EVENT RE.MOVE.TRANSIENT.EFFECT
''
RELEASE LIN.K.USED(*)
RESERVE LIN.K.USED(*) AS LINKS
''
RESET THE TOTALS OF ACTIVE
''
LET CKT.TOTAL = 0
LET CKT.ESTAB = 0
LET CKT.FAILED = 0
LET CKT.DISESTAB = 0
LET HOP.SUM = 0.0
LET HOP.GREATEST = 0.0
LET TOT.HOP.GREATEST = 0
LET HOP.AVG = 0.0
LET DELAY.SUM = 0.0
LET DURATION = 0.0
LET SUM.DURATION = 0.0
LET AVG.DURATION = 0.0
LET LONG.TIME.EST = 0.0
LET AVG.TIME.EST = 0.0
LET AVG.P.BD = 0.0
LET LONG.P.BD = 0.0
LET AVG.C.BD = 0.0
LET LONG.C.BD = 0.0
LET CKT.LONG.TIME.EST = 0
LET AVG.BD.TIME = 0.0
LET SUM.BD.TIME.ALL.CKT = 0.0
LET CKTS.BD = 0
LET P.BD.COUNTER = 0
LET C.BD.COUNTER = 0
LET TOT.P.BD = 0.0
LET TOT.C.BD = 0.0
LET CHANGE.FLAG = 1
LET BACKTRACK.OR.LOOPBACK = 0
LET ACT.LOOP.REMOVE = 0
LET E.SUM = 0.0
LET F.SUB.K.BAR = 0.0
LET FRACT.OF.SUCCESSFUL.CALLS = 0.0
''
PRINT 1 LINE WITH TIME.V AS FOLLOWS
CLEAR COUNTERS AND START TAKING STATS FROM HERE.  TIME.V = ****.******
SKIP 1 OUTPUT LINE
''
RETURN
END ''OF RE.MOVE.TRANSIENT.EFFECT
''
''   THIS EVENT UPDATES THE INFORMATION IN THE BEST.PATH ARRAY BY USE OF
''     THE DIJKSTRA ALGORITHM.  THIS WVWNT IS PERFORMED REGULARLY WITH A
''     PERIOD = "UP.DATE.PERIOD" SECONDS, WHERE THE UP.DATE.PERIOD IS AN
''     INPUT VARIABLE.
''
EVENT DIJK.MANIPULATION
''
DEFINE DIST AS A REAL VARIABLE
LET TOT.DIJK.CALLED = TOT.DIJK.CALLED + 1
''
IF SPECIFY.OUTPUT EQ 0 AND PRT LT 3
  PRINT 2 LINES WITH TIME.V AS FOLLOWS
EVENT DIJK.MANIPULATION INVOKED AT TIME.V = ****.****** SECONDS
-------------------------------------------------------------------
  SKIP 1 OUTPUT LINE
ALWAYS
```

```
''
''   CHECK TO SEE IF A DIJKSTRA UPDATE IS REQUIRED.  A DIJK.MANIPULATION
''     NEED NOT BE PERFORMED IF THERE HAVE BEEN NO CHANGES TO THE SLOT
''     ASSIGNMENTS AT ANY OF THE NODES.
''
IF CHANGE.FLAG EQ 0
  GO TO SCHEDULE      ✓
ALWAYS
''
''   GET THE CURRENT LINK "WEIGHTS" OR "DISTANCES" AT EVERY NODE AND ON
''     ALL LINKS OF THE NETWORK.
''
PERFORM COMPUTE.CURRENT.DISTANCES
''
''   THE PATH.AVAIL ARRAY IS A 2-DIMENSIONAL INTEGER ARRAY THAT HAS ITS
''     VALUES ASSIGNED AND MANIPULATED DURING EACH CALL OF THE DIJK.MA-
''     NIPULATION EVENT.
''
RESERVE PATH.AVAIL(*,*) AS N.NODE BY N.NODE
''
''   USE THE CURRENT NODE AND LINK WEIGHT INFORMATION IN THE IMPLEMENTA-
''     TION OF THE DIJKSTRA ALGORITHM THAT FOLLOWS.  START BY INITIALIZ-
''     ING THE DIJKSTRA AND BEST.PATH ARRAYS.  IF THERE IS NO LINK WHICH
''     DIRECTLY CONNECTS TWO NODES, THEN THE LINK WEIGHT IS SET EQUAL TO
''     999999.9 (OR ANY OTHER LARGE, POSITIVE, REAL NUMBER).  WE MUST
''     ALSO READ A COPY OF THE LINKABLE ARRAY INTO THE PATH.AVAIL ARRAY
''     WHICH WILL BE USED DURING THE DIJK.MANIPULATION EVENT.
''
FOR I = 1 TO N.NODE, DO
  FOR J = 1 TO N.NODE, DO
    IF I NE J AND LINKABLE(I,J) EQ 1
      LET DIJKSTRA(I,J) = DISTANCE(I,J)
      LET BEST.PATH(I,J) = J
      LET PATH.AVAIL(I,J) = 1
      GO TO JUMP.OUT
    ALWAYS
    LET DIJKSTRA(I,J) = 999999.9
    LET BEST.PATH(I,J) = 0
    LET PATH.AVAIL(I,J) = 0
'JUMP.OUT'
  LOOP
LOOP
''
''   PRINT THE INITIAL DIJKSTRA, BEST.PATH AND PATH.AVAIL MATRICES.
''
IF SPECIFY.OUTPUT EQ 0 AND PRT LE 2
  PRINT 5 LINES AS FOLLOWS
THE CONTENTS OF THE INITIAL DIJKSTRA MATRIX ARE:
  +
  +TO         1          2          3          4          5          6          7
FROM+
----+--------------------------------------------------------------------------------
  FOR I = 1 TO N.NODE, DO
    PRINT 1 LINE WITH I, DIJKSTRA(I,1), DIJKSTRA(I,2), DIJKSTRA(I,3),
      DIJKSTRA(I,4), DIJKSTRA(I,5), DIJKSTRA(I,6) AND DIJKSTRA(I,7)
      AS FOLLOWS
 **     ******.* ******.* ******.* ******.* ******.* ******.* ******.*
  LOOP
SKIP 1 OUTPUT LINE
  PRINT 5 LINES AS FOLLOWS
  INITIAL DIJKSTRA MATRIX (CONT.):
  +
  +TO         8          9          10         11         12         13
FROM+
----+--------------------------------------------------------------------------------
  FOR I = 1 TO N.NODE, DO
    PRINT 1 LINE WITH I, DIJKSTRA(I,8), DIJKSTRA(I,9), DIJKSTRA(I,10),
      DIJKSTRA(I,11), DIJKSTRA(I,12) AND DIJKSTRA(I,13) AS FOLLOWS
 **   + ******.* ******.* ******.* ******.* ******.* ******.*
  LOOP
  SKIP 2 OUTPUT LINES
  PRINT 5 LINES AS FOLLOWS
```

```
THE CONTENTS OF THE INITIAL BEST.PATH ARRAY ARE:
   +
   +TO    1     2     3     4     5     6     7     8     9    10    11    12    13
FROM+
-----+-----------------------------------------------------------------------------
   FOR I = 1 TO N.NODE, DO
      PRINT 1 LINE WITH I, BEST.PATH(I,1), BEST.PATH(I,2), BEST.PATH(I,3),
         BEST.PATH(I,4), BEST.PATH(I,5), BEST.PATH(I,6), BEST.PATH(I,7),
         BEST.PATH(I,8), BEST.PATH(I,9), BEST.PATH(I,10), BEST.PATH(I,11),
         BEST.PATH(I,12) AND BEST.PATH(I,13) AS FOLLOWS
 **  +  **    **    **    **    **    **    **    **    **    **    **    **    **
   LOOP
   SKIP 2 OUTPUT LINES
''
   PRINT 5 LINES AS FOLLOWS
THE CONTENTS OF THE INITIAL PATH.AVAIL ARRAY ARE:
   +
   +TO    1     2     3     4     5     6     7     8     9    10    11    12    13
FROM+
-----+-----------------------------------------------------------------------------
   FOR I = 1 TO N.NODE, DO
      PRINT 1 LINE WITH I, PATH.AVAIL(I,1), PATH.AVAIL(I,2),
         PATH.AVAIL(I,3), PATH.AVAIL(I,4), PATH.AVAIL(I,5),
         PATH.AVAIL(I,6), PATH.AVAIL(I,7), PATH.AVAIL(I,8),
         PATH.AVAIL(I,9), PATH.AVAIL(I,10), PATH.AVAIL(I,11),
         PATH.AVAIL(I,12) AND PATH.AVAIL(I,13) AS FOLLOWS
 **  +  **    **    **    **    **    **    **    **    **    **    **    **    **
   LOOP
   SKIP 2 OUTPUT LINES
''
REGARDLESS
LET MANIP.COUNTER = 0
LET PASS.COUNTER = 0
''
'RUN.MATRIX.AGAIN'
LET AGAIN.FLAG = 0
LET PASS.COUNTER = PASS.COUNTER + 1
FOR ROW = 1 TO N.NODE, DO
   FOR COL = 1 TO N.NODE, DO
      IF ROW EQ COL
         GO TO NEXT.COL
      ELSE
         FOR TEST.COL = 1 TO N.NODE, DO
            IF TEST.COL EQ ROW
               GO TO NEXT.TEST.COL
            ELSE
               IF TEST.COL EQ COL
                  GO TO NEXT.TEST.COL
               ELSE
                  LET DIST = 0.0
                  IF LINKABLE(ROW,TEST.COL) EQ 1
                     LET DIST = DIJKSTRA(ROW,TEST.COL)
                     IF PATH.AVAIL(TEST.COL,COL) EQ 1
                        LET DIST = DIST + DIJKSTRA(TEST.COL,COL)
                        IF DIST LT DIJKSTRA(ROW,COL)
                           LET DIJKSTRA(ROW,COL) = DIST
                           LET BEST.PATH(ROW,COL) = BEST.PATH(ROW,TEST.COL)
                           LET PATH.AVAIL(ROW,COL) = 1
                           LET AGAIN.FLAG = 1
                           LET MANIP.COUNTER = MANIP.COUNTER + 1
                        REGARDLESS
                     REGARDLESS
                  REGARDLESS
'NEXT.TEST.COL'
         LOOP
'NEXT.COL'
   LOOP
LOOP
''
IF AGAIN.FLAG EQ 1
   GO TO RUN.MATRIX.AGAIN
ALWAYS
```

```
''
''   WE MIGHT NOW WANT TO PRINT THE MANIPULATED DIJKSTRA AND BEST.PATH
''     MATRICES.
''
IF SPECIFY.OUTPUT EQ 0 AND PRT LE 2
  PRINT 2 LINES WITH PASS.COUNTER AND MANIP.COUNTER AS FOLLOWS
WE MADE **** PASSES THROUGH THE DIJKSTRA ARRAY AND PERFORMED A TOTAL
  OF ***** MANIPULATIONS IN DETERMINING THE NEW BEST PATH NEIGHBORS.
  SKIP 1 OUTPUT LINE
''
  PRINT 5 LINES AS FOLLOWS
THE CONTENTS OF THE MANIPULATED DIJKSTRA ARRAY ARE:
 +
  +TO          1          2          3          4          5          6          7
FROM+
-----+----------------------------------------------------------------------------

  FOR I = 1 TO N.NODE, DO
    PRINT 1 LINE WITH I, DIJKSTRA(I,1), DIJKSTRA(I,2), DIJKSTRA(I,3),
      DIJKSTRA(I,4), DIJKSTRA(I,5), DIJKSTRA(I,6) AND DIJKSTRA(I,7)
      AS FOLLOWS
 **   + ******.* ******.* ******.* ******.* ******.* ******.* ******.*
  LOOP
  SKIP 1 OUTPUT LINE
  PRINT 5 LINES AS FOLLOWS
  MANIPULATED DIJKSTRA ARRAY (CONT.):
  +
  +TO          8          9          10         11         12         13
FROM+
-----+----------------------------------------------------------------------------
  FOR I = 1 TO N.NODE, DO
    PRINT 1 LINE WITH I, DIJKSTRA(I,8), DIJKSTRA(I,9), DIJKSTRA(I,10),
      DIJKSTRA(I,11), DIJKSTRA(I,12) AND DIJKSTRA(I,13) AS FOLLOWS
 **   + ******.* ******.* ******.* ******.* ******.* ******.*
  LOOP
  SKIP 2 OUTPUT LINES
''
  PRINT 5 LINES AS FOLLOWS
THE CONTENTS OF THE MANIPULATED BEST.PATH ARRAY ARE:
 +
  +TO    1    2    3    4    5    6    7    8    9   10   11   12   13
FROM+
-----+----------------------------------------------------------------------------
  FOR I = 1 TO N.NODE, DO
    PRINT 1 LINE WITH I, BEST.PATH(I,1), BEST.PATH(I,2), BEST.PATH(I,3),
      BEST.PATH(I,4), BEST.PATH(I,5), BEST.PATH(I,6), BEST.PATH(I,7),
      BEST.PATH(I,8), BEST.PATH(I,9), BEST.PATH(I,10), BEST.PATH(I,11),
      BEST.PATH(I,12) AND BEST.PATH(I,13) AS FOLLOWS
 **   +  **   **   **   **   **   **   **   **   **   **   **   **   **
  LOOP
  SKIP 2 OUTPUT LINES
''
  PRINT 5 LINES AS FOLLOWS
THE CONTENTS OF THE MANIPULATED PATH.AVAIL ARRAY ARE:
 +
  +TO    1    2    3    4    5    6    7    8    9   10   11   12   13
FROM+
-----+----------------------------------------------------------------------------
  FOR I = 1 TO N.NODE, DO
    PRINT 1 LINE WITH I, PATH.AVAIL(I,1), PATH.AVAIL(I,2),
      PATH.AVAIL(I,3), PATH.AVAIL(I,4), PATH.AVAIL(I,5),
      PATH.AVAIL(I,6), PATH.AVAIL(I,7), PATH.AVAIL(I,8),
      PATH.AVAIL(I,9), PATH.AVAIL(I,10), PATH.AVAIL(I,11),
      PATH.AVAIL(I,12) AND PATH.AVAIL(I,13) AS FOLLOWS
 **   +  **   **   **   **   **   **   **   **   **   **   **   **   **
  LOOP
  SKIP 2 OUTPUT LINES
''
ALWAYS
''
''   SCHEDULE THE NEXT DIJK.MANIPULATION.
''
```

```
'SCHEDULE'
LET CHANGE.FLAG = 0
SCHEDULE A DIJK.MANIPULATION IN UP.DATE.PERIOD UNITS
''
RETURN
END  ''OF DIJK.MANIPULATION
''
''   THIS ROUTINE IS CALLED BY THE DIJK.MANIPULATION EVENT TO DETERMINE
''     THE CURRENT LINK DISTANCES (A.K.A WEIGHTS OR CHANNEL VALUES) FOR
''     EACH NODE ON EACH DIRECT LINK.  THIS ROUTINE WILL USE A "DISTANCE
''     FUNCTION" TO EVALUATE THE LINK WEIGHTS.  THE DISTANCE FUNCTION
''     WILL BE CHANGED MANY TIMES THROUGHOUT THE COURSE OF THE THESIS
''     RESEARCH AS WE INVESTIGATE THE EFFECTS OF THE DISTANCE FUNCTION
''     OF NETWORK ROUTING, CAPACITY AND THROUGHPUT.
''
ROUTINE TO COMPUTE.CURRENT.DISTANCES
''
DEFINE X AND Y AS REAL VARIABLES
DEFINE WEIGHT AS A REAL VARIABLE
''
''   WE MAY WANT TO USE THE NUMBER OF NEIGHBOR NODES CLAIMED BY A NODE
''     AS A TERM OR CONSIDERATION WHEN WE COMPUTE THE "NODE WEIGHT" FAC-
''     TOR OF AN OVERALL LINK WEIGHT.  THE NUMBER OF NEIGHBOR NODES
''     CLAIMED BY EACH NODE N HAS ALREADY BEEN DETERMINED AND HAS BEEN
''     STORED IN LINKABLE(N,N).
''
''   COMPUTE "NODE WEIGHTS" FOR EACH NODE AND STORE IN DISTANCE(I,I).
''
LET X = REAL.F(SLOTS * MAX.SLOT.DEPTH) * 2.0
FOR A = 1 TO N.NODE, DO
   FOR B = A TO N.NODE, DO
      IF A NE B AND LINKABLE(A,B) EQ 1
         LET SUM = 0
         FOR K = 1 TO SLOTS, DO
            IF USE(A,K,1) EQ 0 AND USE(A,K,4) EQ 0 AND USE(B,K,1) EQ 0 AND
               USE(B,K,4) EQ 0
               LET SUM = SUM + (2 * MAX.SLOT.DEPTH)
               GO TO LOOOP
            ALWAYS
            IF USE(A,K,1) EQ 0 AND USE(A,K,4) EQ 0 AND USE(B,K,1) EQ 0 AND
               USE(B,K,4) NE 0
               LET SUM = SUM + (MAX.SLOT.DEPTH - USE(B,K,4))
               GO TO LOOOP
            ALWAYS
            IF USE(A,K,1) EQ 0 AND USE(A,K,4) NE 0 AND USE(B,K,1) EQ 0 AND
               USE(B,K,4) EQ 0
               LET SUM = SUM + (MAX.SLOT.DEPTH - USE(A,K,4))
               GO TO LOOOP
            ALWAYS
'LOOOP'
         LOOP
         LET WEIGHT = X - REAL.F(SUM)
         LET Y = (WEIGHT * 128.0) / X
         IF INT.F(Y) EQ 0
            LET Y = 1.0
         ALWAYS
         LET DISTANCE(A,B) = NODE.SCALE(INT.F(Y))
         LET DISTANCE(B,A) = NODE.SCALE(INT.F(Y))
      ALWAYS
   LOOP
LOOP
''
''   WE HAVE NOW STORED THE NODE WEIGHT VALUE IN THE DISTANCE ARRAY.
''
''   WE MIGHT WANT TO PRINT THE DISTANCE ARRAY NOW TO ENSURE THAT THE
''     NODE WEIGHTS WERE PROPERLY CALCULATED AND RECORDED.
''
IF SPECIFY.OUTPUT EQ 0 AND PRT LT 3
   PRINT 6 LINES AS FOLLOWS
THE CONTENTS OF THE DISTANCE ARRAY AFTER THE NODE WEIGHTS WERE
   CALCULATED ARE:
   +
```

```
   +TO          1          2          3          4          5          6          7
FROM+
-----+-----------------------------------------------------------------------------
   FOR I = 1 TO N.NODE, DO
      PRINT 1 LINE WITH I, DISTANCE(I,1), DISTANCE(I,2), DISTANCE(I,3),
         DISTANCE(I,4), DISTANCE(I,5), DISTANCE(I,6) AND DISTANCE(I,7)
         AS FOLLOWS
 **   + ******.* ******.* ******.* ******.* ******.* ******.* ******.*
   LOOP
   SKIP 1 OUTPUT LINE
   PRINT 5 LINES AS FOLLOWS
   CONTENTS OF THE DISTANCE ARRAY (CONT.):
   +
   +TO          8          9          10         11         12         13
FROM+
-----+-----------------------------------------------------------------------------
   FOR I = 1 TO N.NODE, DO
      PRINT 1 LINE WITH I, DISTANCE(I,8), DISTANCE(I,9), DISTANCE(I,10),
         DISTANCE(I,11), DISTANCE(I,12) AND DISTANCE(I,13) AS FOLLOWS
 **   + ******.* ******.* ******.* ******.* ******.* ******.*
   LOOP
   SKIP 2 OUTPUT LINES
ALWAYS
''
''    WE CAN NOW MODIFY THE NODE WEIGHTS JUST CALCULATED TO PRODUCE A
''       TRUE LINK WEIGHT BY ADDING THE LINK WEIGHT FROM THE APPROPRIATE
''       ENTRY IN THE "LINK.WEIGHT" ARRAY.  RECALL THAT THESE LINK WEIGHTS
''       WERE CALCULATED IN THE "HOUSEKEEPING" ROUTINE.  LINK ATTENUATIONS
''       RANGED IN VALUE FROM ABOUT 81.0 TO 141.0 DB (I.E. A RANGE OF
''       ABOUT 60 DB).  THE ENERGY PER BIT FOR THESE LINKS RANGED IN VALUE
''       FROM ABOUT 1.0 TO 1000000.0 AND WERE SCALED WITH A GEOMETRIC DIS-
''       TRIBUTION INTO "BINS" WITH ASSIGNED LINK WEIGHTS OF 1.0 TO 128.0.
''
FOR A = 1 TO N.NODE, DO
   FOR B = 1 TO N.NODE, DO
      IF A NE B AND LINKABLE(A,B) EQ 1
         LET DISTANCE(A,B) = DISTANCE(A,B) + LINK.WEIGHT(A,B)
         GO TO GET.AWAY
      ALWAYS
      LET DISTANCE(A,B) = 999999.9
'GET.AWAY'
   LOOP
LOOP
''
''    WE MIGHT WANT TO PRINT THE DISTANCE ARRAY NOW TO ENSURE THAT THE
''       OVERALL LINK WEIGHTS WERE PROPERLY CALCULATED AND RECORDED.
''
IF SPECIFY.OUTPUT EQ 0 AND PRT LT 3
   PRINT 6 LINES AS FOLLOWS
THE CONTENTS OF THE DISTANCE ARRAY AFTER THE LINK WEIGHTS WERE
   CALCULATED ARE:
   +
   +TO          1          2          3          4          5          6          7
FROM+
-----+-----------------------------------------------------------------------------
   FOR I = 1 TO N.NODE, DO
      PRINT 1 LINE WITH I, DISTANCE(I,1), DISTANCE(I,2), DISTANCE(I,3),
         DISTANCE(I,4), DISTANCE(I,5), DISTANCE(I,6) AND DISTANCE(I,7)
         AS FOLLOWS
 **   + ******.* ******.* ******.* ******.* ******.* ******.* ******.*
   LOOP
   SKIP 1 OUTPUT LINE
   PRINT 5 LINES AS FOLLOWS
   DISTANCE ARRAY LINK WEIGHTS (CONT.):
   +
   +TO          8          9          10         11         12         13
FROM+
-----+-----------------------------------------------------------------------------
   FOR I = 1 TO N.NODE, DO
      PRINT 1 LINE WITH I, DISTANCE(I,8), DISTANCE(I,9), DISTANCE(I,10),
         DISTANCE(I,11), DISTANCE(I,12) AND DISTANCE(I,13) AS FOLLOWS
 **   + ******.* ******.* ******.* ******.* ******.* ******.*
```

```
   LOOP
   SKIP 2 OUTPUT LINES
ALWAYS
''
RETURN
END ''OF COMPUTE.CURRENT.DISTANCES
''    THIS EVENT PERFORMS FUNCTIONS NECESSARY TO BEGIN PROCESSING NEW
''       REQUIREMENTS FOR TWO-WAY VIRTUAL VOICE CIRCUITS.
''
EVENT NEW.CKT.REQMT
''
IF PRNT LE 1
   PRINT 2 LINES WITH TIME.V AS FOLLOWS
EVENT NEW.CKT.REQMT INVOKED AT TIME.V = ****.******
-----------------------------------------------------------
   SKIP 1 OUTPUT LINE
ALWAYS
DEFINE CK.XMTR,CK.RCVR,X.TOT.PERCENT AND R.TOT.PERCENT AS REAL VARIABLES
DEFINE DELAY1 AS A REAL VARIABLE
LET CKT.TOTAL = CKT.TOTAL + 1
LET CKT.SUM = CKT.SUM + 1
IF CKT.SUM GT MAX.CKTS.IN.SIM
   SKIP 2 OUTPUT LINES
   PRINT 12 LINES WITH MAX.SLOT.DEPTH, MAX.CKTS.IN.SIM, TEST.DURATION AND
      TIME.V AS FOLLOWS
TOTAL NUMBER OF CIRCUITS ATTEMPTED EXCEEDS THE TOTAL NUMBER OF CIRCUITS
   PERMITTED.  IN THE FUTURE IF WE WANT THE SIMULATION FOR THIS VALUE OF
   SLOT.DEPTH = ** TO RUN FOR THE COMPLETE SIMULATION TEST.DURATION, WE
   MUST DO ONE OF THE FOLLOWING:

      1.  INCREASE MAX.CKTS.IN.SIM FROM ITS PRESENT VALUE OF *****
      2.  DECREASE THE SIMULATION TEST.DURATION FROM ITS PRESENT VALUE OF
             *****.****** SECONDS,
      3.  OR DO SOME COMBINATION OF BOTH 1 AND 2 ABOVE.

SIMULATION TIME AT THE INSTANT EXECUTION WAS HALTED = *****.****** SEC.
-----------------------------------------------------------------------
   PERFORM DESTRUCTION
   GO TO RTN
REGARDLESS
''
''    SCHEDULE THE NEXT "NEW.CKT.REQMT" EVENT FOR THE NETWORK.        rate
''
SCHEDULE A NEW.CKT.REQMT IN EXPONENTIAL.F(MEAN.CKT.ESTAB,2) UNITS
''
''    FIND A DESTINATION NODE IN ACCORDANCE WITH PRESCRIBED RECEIVE PER-
''       CENTS FOR THE NODES.
''
LET X.TOT.PERCENT = 0.0   l1  l2   1
LET Y.TOT.PERCENT = 0.0
LET CK.XMTR = UNIFORM.F(0.0,TRNS.PCNT,6)
''
''    SELECTOR IS USED IF A PERCENTAGE OF THE MESSAGES ARE REQUIRED TO BE
''       BETWEEN NODES OF THE SAME GROUP OR FAMILY.
''
LET SELECTOR = UNIFORM.F(0.0,100.,7)
''
''    SELECT THE TRANSMITTING NODE.
''
FOR I = 1 TO N.NODE, DO
   LET X.TOT.PERCENT = X.TOT.PERCENT + TRANSMIT.PERCENT(I)
   IF CK.XMTR LE X.TOT.PERCENT
      LET XMTR = I
      GO FIND.RECEIVER
   ELSE
LOOP
''
''    SELECT THE RECEIVER.
''
'FIND.RECEIVER'
LET CK.RCVR = UNIFORM.F(0.0,RCV.PCNT,8)
FOR J = 1 TO N.NODE, DO
```

```
   LET R.TOT.PERCENT = R.TOT.PERCENT + RECEIVE.PERCENT(J)
   IF CK.RCVR LE R.TOT.PERCENT
      LET RCVR = J
      GO CK.GROUPS.AND.FAMILIES
   ELSE
LOOP
''
''    IF THE RECEIVER MUST BE WITHIN THE SAME GROUP OR FAMILY, KEEP
''       LOOKING UNTIL AN ADEQUATE RECEIVER IS FOUND.
''
'CK.GROUPS.AND.FAMILIES'
IF SELECTOR LT IN.GROUP
   IF GROUP(XMTR) EQ GROUP(RCVR)
      GO SEE.IF.XMTR.EQ.RCVR
   ELSE
      LET R.TOT.PERCENT = 0.0
      GO FIND.RECEIVER
ELSE
IF SELECTOR LT (IN.GROUP + IN.FAMILY)
   IF FAMILY(XMTR) EQ FAMILY(RCVR)
      GO SEE.IF.XMTR.EQ.RCVR
   ELSE
      LET R.TOT.PERCENT = 0.0
      GO FIND.RECEIVER
ELSE
''
'SEE.IF.XMTR.EQ.RCVR'
IF RCVR EQ XMTR
   GO FIND.RECEIVER
ELSE
LET ORIG.NODE = XMTR
LET DEST.NODE = RCVR
IF PRNT LE 4
   PRINT 1 LINE WITH CKT.SUM, ORIG.NODE, DEST.NODE AND TIME.V AS FOLLOWS
CIRCUIT NR. *****, FROM NODE ** TO NODE ** BEGUN AT TIME = ****.******
   SKIP 1 OUTPUT LINE
ALWAYS
''
''    WE CAN NOW BEGIN TO ESTABLISH THE CIRCUIT.  THE REMAINDER OF THIS
''       EVENT SIMULATES ALL OF THE ACTIONS PERFORMED AT THE ORIGINATING
''       NODE TO GENERATE AND TRANSMIT THE SERVICE OR COORDINATION MESSAGE
''       TO THE NEXT NODE (I.E. THE "CALLED.NODE") ON THE BEST PATH TO THE
''       DESTINATION NODE.
''
''    FIRST CHECK TO SEE IF THERE IS A SLOT AVAILABLE AT THE ORIG.NODE TO
''       ACCOMODATE THE TRANSMISSION OF A SERVICE MESSAGE.  SINCE WE ARE
''       ASSUMING THAT EACH NODE IS ALWAYS LISTENING TO ITS NEIGHBORS, THE
''       ORIG.NODE KNOWS WHEN ITS NEIGHBORS ARE NOT TRANSMITTING.  NOTE:
''       ALL NODES "LISTEN" WHENEVER THEY ARE NOT TRANSMITTING.
''
LET UP.ROUTE = UP.ROUTE + 1
LET CALLED.NODE = BEST.PATH(ORIG.NODE,DEST.NODE)
FOR J = 1 TO SLOTS, DO
   IF USE(ORIG.NODE,J,1) EQ 0 AND USE(ORIG.NODE,J,4) EQ 0 AND
      USE(CALLED.NODE,J,1) EQ 0
      GO TO PASS1
   ELSE
LOOP
IF PRNT LE 4
   PRINT 4 LINES WITH CKT.SUM, ORIG.NODE AND CALLED.NODE AS FOLLOWS
CIRCUIT NR. ***** FROM NODE ** TO NODE **.  THIS CIRCUIT CANNOT BE
   ESTABLISHED BECAUSE THERE ARE NO MUTUALLY AVAILABLE SLOTS BETWEEN THE
   THE ORIG.NODE AND THE CALLED.NODE TO CARRY THE INITIAL SERVICE MES-
   SAGE.
   SKIP 1 OUTPUT LINE
ALWAYS
LET CKT.FAILED = CKT.FAILED + 1
LET UP.ROUTE = UP.ROUTE - 1
LET P.BD.COUNTER = P.BD.COUNTER + 1
GO TO RTN
''
''    RANDOMLY SELECT A "CURRENT.SLOT" AND CONTINUE PROCESSING.
```

```
''
'PASS1'
LET CURRENT.SLOT = RANDI.F(1,SLOTS,4)
IF PRNT LE 1
   PRINT 2 LINES WITH CURRENT.SLOT AS FOLLOWS
   SLOT ** WAS RANDOMLY SELECTED AS THE "CURRENT.SLOT" AS WE BEGAN ESTAB-
     LISHING THE CIRCUIT IN THE EVENT NEW.CKT.REQMT.
   SKIP 1 OUTPUT LINE
ALWAYS
''
''    FIND THE NEXT MUTUALLY AVAILABLE SLOT (AT LEAST 1 FULL SLOT IN THE
''      FUTURE TO ACCOUNT FOR PROCESSING TIME IN THE ORIG.NODE).
''
LET SLOT1 = 0
LET FRAME1 = 0
IF CURRENT.SLOT EQ (SLOTS - 1)
   LET K = 1
   GO TO SEARCH.NEXT.FRAME
ALWAYS
IF CURRENT.SLOT EQ SLOTS
   LET K = 2
   GO TO SEARCH.NEXT.FRAME
ALWAYS
LET K = CURRENT.SLOT + 2
FOR J = K TO SLOTS, DO
   IF USE(ORIG.NODE,J,1) EQ 0 AND USE(ORIG.NODE,J,4) EQ 0 AND
     USE(CALLED.NODE,J,1) EQ 0
     LET SLOT1 = J
     GO TO PASS2
   ALWAYS
LOOP
LET K = 1
''
'SEARCH.NEXT.FRAME'
LET FRAME1 = 1
FOR J = K TO SLOTS, DO
   IF USE(ORIG.NODE,J,1) EQ 0 AND USE(ORIG.NODE,J,4) EQ 0 AND
     USE(CALLED.NODE,J,1) EQ 0
     LET SLOT1 = J
     GO TO PASS2
   ALWAYS
LOOP
IF USE(ORIG.NODE,1,1) EQ 0 AND USE(ORIG.NODE,1,4) EQ 0 AND
   USE(CALLED.NODE,1,1) EQ 0
   LET FRAME1 = 2
   LET SLOT1 = 1
   GO TO PASS2
ALWAYS
PRINT 1 LINE WITH CKT.SUM AS FOLLOWS
INITIAL SVC MSG IN ERROR IN EVENT NEW.CKT.REQMT FOR CIRCUIT NR. *****
SKIP 1 OUTPUT LINE
LET CKT.FAILED = CKT.FAILED + 1
LET UP.ROUTE = UP.ROUTE - 1
LET P.BD.COUNTER = P.BD.COUNTER + 1
GO TO RTN
''
''    IF WE GET AS FAR AS PASS2 THEN WE HAVE IDENTIFIED A SLOT TO CARRY
''      THE SERVICE MESSAGE TO THE CALLED.NODE.  NOW CREATE THE SERVICE
''      MESSAGE.
''
'PASS2'
CREATE A MESSAGE
   LET CKT.NR(MESSAGE)= CKT.SUM
   LET TYPE(MESSAGE) = PACKET
   LET ORIGINATOR(MESSAGE) = ORIG.NODE
   LET DESTINATION(MESSAGE) = DEST.NODE
   LET FM.NODE(MESSAGE) = ORIG.NODE
   LET TO.NODE(MESSAGE) = CALLED.NODE
   LET START.TIME(MESSAGE) = TIME.V
   LET HOP.COUNT(MESSAGE) = 0.0
   LET SLOT.ARRIVAL(MESSAGE) = SLOT1
   LET SLOT.ASSIGN(MESSAGE) = 0
```

```
   LET RECSLCT(MESSAGE) = 0
   LET DIRECTION(MESSAGE) = 0
   LET CUM.ENERGY(MESSAGE) = 0.0
   LET INFO1(MESSAGE) = 0
   LET INFO2(MESSAGE) = 0
   LET INFO3(MESSAGE) = 0
   LET INFO4(MESSAGE) = 0
   LET INFO5(MESSAGE) = 0
   LET INFO6(MESSAGE) = 0
   LET INFO7(MESSAGE) = 0
   LET INFO8(MESSAGE) = 0
   LET INFO9(MESSAGE) = 0
IF PRNT LE 1
   PRINT 2 LINES WITH SLOT1 AND FRAME1 AS FOLLOWS
   SLOT ** OF FRAME * WAS SELECTED TO CARRY THE INITIAL REQUEST FOR SER-
     VICE FROM THE ORIG.NODE TO THE CALLED.NODE.
   SKIP 1 OUTPUT LINE
ALWAYS
''
''   CALCULATE WHEN THE SERVICE MESSAGE WILL ARRIVE AT THE CALLED.NODE
''      AND SCHEDULE ITS ARRIVAL IN TIME.V PLUS THAT INCREMENT.
''
IF FRAME1 EQ 0
   LET DELAY1 = (REAL.F(SLOT1 - CURRENT.SLOT)) * SLOT.DURATION
   GO TO PASS3
ALWAYS
IF FRAME1 EQ 1
   LET X = ((SLOTS + 1) - CURRENT.SLOT)
   LET Y = SLOT1 - 1
   LET DELAY1 = (REAL.F(X + Y)) * SLOT.DURATION
   GO TO PASS3
ALWAYS
IF FRAME1 EQ 2
   LET DELAY1 = (REAL.F(SLOTS + 1)) * SLOT.DURATION
   GO TO PASS3
ALWAYS
PRINT 1 LINE WITH CKT.SUM AS FOLLOWS
ERROR IN CALCULATING DELAY1 IN EVENT NEW.CKT.REQMT.  CKT.TOTAL = *****
SKIP 1 OUTPUT LINE
LET CKT.FAILED = CKT.FAILED + 1
LET UP.ROUTE = UP.ROUTE - 1
LET P.BD.COUNTER = P.BD.COUNTER + 1
DESTROY THE MESSAGE CALLED MESSAGE
GO TO RTN
''
'PASS3'
SCHEDULE AN INITIAL.REQ.FOR.SVC GIVEN MESSAGE IN DELAY1 UNITS
IF PRNT LE 1
   PRINT 2 LINES WITH CKT.SUM, CALLED.NODE, (TIME.V + DELAY1) AND
     DELAY1 AS FOLLOWS
   CIRCUIT NR. ***** HAS SCHEDULED AN INITIAL.REQ.FOR.SVC AT NODE ** AT
     TIME.V = ****.****** SECONDS, I.E. *.****** SECONDS FROM NOW.
   SKIP 2 OUTPUT LINES
ALWAYS
''
'RTN'
IF PRNT LE 1
   PRINT 1 LINE AS FOLLOWS
   ATTRIBUTES OF THE MESSAGE ENTITY AT THE END OF NEW.CKT.REQMT ARE:
   LIST ATTRIBUTES OF MESSAGE
   SKIP 1 OUTPUT LINE
ALWAYS
RETURN
END ''OF NEW.CKT.REQMT
''
''   THIS EVENT SIMULATES THE INITIAL REQUEST FOR SERVICE FROM A
''      CALLING.NODE TO A CALLED.NODE.  THE PROCESSING DONE HEREIN IS
''      DONE AT THE CALLED.NODE.
''
EVENT INITIAL.REQ.FOR.SVC GIVEN SVC1.MSG
LET MESSAGE = SVC1.MSG
IF PRNT LE 1
```

```
   PRINT 2 LINES WITH TIME.V AS FOLLOWS
EVENT INITIAL.REQ.FOR.SVC INVOKED AT TIME.V = ****.******
-----------------------------------------------------------------
   SKIP 1 OUTPUT LINE
ALWAYS
IF PRNT LE 3
   PRINT 1 LINE AS FOLLOWS
   ATTRIBUTES OF MESSAGE ENTITY AT THE START OF INITIAL.REQ.FOR.SVC ARE:
   LIST ATTRIBUTES OF MESSAGE
   SKIP 1 OUTPUT LINE
ALWAYS
DEFINE DELAY2 AS A REAL VARIABLE
LET FRAME.REC = 0
LET SLOT.REC = 0
LET CALLING.NODE = FM.NODE(MESSAGE)
LET CALLED.NODE = TO.NODE(MESSAGE)
''
''    FIRST CHECK TO SEE IF THIS CALLED.NODE ALREADY HAS SLOTS ASSIGNED
''       TO CARRY THIS CIRCUIT NUMBER.  IF IT DOES, THEN THE CIRCUIT HAS
''       BACKTRACKED OR LOOPED BACK ACROSS ITSELF AS A RESULT OF CHANGES
''       TO THE BEST PATH ROUTE AS DETERMINED BY THE DIJK.MANIPULATION
''       EVENT, AND WE MUST REMOVE THE SLOT ASSIGNMENTS IN THE LOOP SINCE
''       THEY ARE NO LONGER NECESSARY.
''
FOR I = 1 TO SLOTS, DO
   IF USE(CALLED.NODE,I,1) EQ CKT.NR(MESSAGE)
      LET BACKTRACK.OR.LOOPBACK = BACKTRACK.OR.LOOPBACK + 1
      LET ACT.LOOP.REMOVE = ACT.LOOP.REMOVE + 1
      CREATE A MESSAGE CALLED LOOP.BD.MSG
         LET CKT.NR(LOOP.BD.MSG) = CKT.NR(MESSAGE)
         LET TYPE(LOOP.BD.MSG) = REMOVE.LOOP
         LET ORIGINATOR(LOOP.BD.MSG) = CALLED.NODE
         LET DESTINATION(LOOP.BD.MSG) = CALLING.NODE
         LET FM.NODE(LOOP.BD.MSG) = TO.NODE(MESSAGE)
         LET TO.NODE(LOOP.BD.MSG) = TO.NODE(MESSAGE)
         LET START.TIME(LOOP.BD.MSG) = TIME.V
         LET HOP.COUNT(LOOP.BD.MSG) = HOP.COUNT(MESSAGE)
         LET SLOT.ARRIVAL(LOOP.BD.MSG) = SLOT.ARRIVAL(MESSAGE)
         LET SLOT.ASSIGN(LOOP.BD.MSG) = SLOT.ASSIGN(MESSAGE)
         LET RECSLOT(LOOP.BD.MSG) = RECSLOT(MESSAGE)
         LET DIRECTION(LOOP.BD.MSG) = -2
         LET CUM.ENERGY(LOOP.BD.MSG) = 0.0
         LET INFO1(LOOP.BD.MSG) = INFO1(MESSAGE)
         LET INFO2(LOOP.BD.MSG) = INFO2(MESSAGE)
         LET INFO3(LOOP.BD.MSG) = INFO3(MESSAGE)
         LET INFO4(LOOP.BD.MSG) = INFO4(MESSAGE)
         LET INFO5(LOOP.BD.MSG) = INFO5(MESSAGE)
         LET INFO6(LOOP.BD.MSG) = INFO6(MESSAGE)
         LET INFO7(LOOP.BD.MSG) = INFO7(MESSAGE)
         LET INFO8(LOOP.BD.MSG) = INFO8(MESSAGE)
         LET INFO9(LOOP.BD.MSG) = INFO9(MESSAGE)
''
''    WE HAVE CREATED ANOTHER MESSAGE TO SEND IN THE UPSTREAM DIRECTION
''       TO REMOVE THE SLOT ASSIGNMENTS AT THE NODES IN THE LOOP.  NOTE
''       HOWEVER THAT IF WE HAVE LOOPED BACK THROUGH THE ORIGINATOR NODE
''       THEN WE MUST CREATE A MESSAGE TO CONTINUE THE CIRCUIT ESTABLISH-
''       MENT BEFORE WE SCHEDULE AN UPSTREAM.BREAK.DOWN TO DESTROY THE
''       LOOP.
''
      FOR J = 1 TO SLOTS, DO
         IF USE(CALLED.NODE,J,1) EQ CKT.NR(MESSAGE) AND
            USE(CALLED.NODE,J,5) NE 0
            GO TO MAKE.A.MESSAGE
         ALWAYS
      LOOP
      FOR J = 1 TO SLOTS, DO
         IF USE(CALLED.NODE,J,1) EQ CKT.NR(MESSAGE) AND
            USE(CALLED.NODE,J,5) EQ 0
               GO TO MAKE.A.MESSAGE
         ALWAYS
      LOOP
''
```

Drop

```
'MAKE.A.MESSAGE'
    CREATE A MESSAGE CALLED CONT.MSG
      LET CKT.NR(CONT.MSG) = CKT.NR(MESSAGE)
      LET TYPE(CONT.MSG) = TYPE(MESSAGE)
      LET ORIGINATOR(CONT.MSG) = ORIGINATOR(MESSAGE)
      LET DESTINATION(CONT.MSG) = DESTINATION(MESSAGE)
      LET FM.NODE(CONT.MSG) = CALLED.NODE
      LET TO.NODE(CONT.MSG) = BEST.PATH(CALLED.NODE,
        DESTINATION(MESSAGE))
      LET START.TIME(CONT.MSG) = START.TIME(MESSAGE)
      LET HOP.COUNT(CONT.MSG) = REAL.F(USE(CALLED.NODE,J,5))
      LET SLOT.ARRIVAL(CONT.MSG) = SLOT.ARRIVAL(MESSAGE)
      LET SLOT.ASSIGN(CONT.MSG) = SLOTS + 1
      LET RECSLOT(CONT.MSG) = 0
      LET DIRECTION(CONT.MSG) = DIRECTION(MESSSAGE)
      LET CUM.ENERGY(CONT.MSG) = REAL.F(USE(CALLED.NODE,J,6))
      LET INFO1(CONT.MSG) = INFO1(MESSAGE)
      LET INFO2(CONT.MSG) = INFO2(MESSAGE)
      LET INFO3(CONT.MSG) = INFO3(MESSAGE)
      LET INFO4(CONT.MSG) = INFO4(MESSAGE)
      LET INFO5(CONT.MSG) = INFO5(MESSAGE)
      LET INFO6(CONT.MSG) = INFO6(MESSAGE)
      LET INFO7(CONT.MSG) = INFO7(MESSAGE)
      LET INFO8(CONT.MSG) = INFO8(MESSAGE)
      LET INFO9(CONT.MSG) = INFO9(MESSAGE)
    SCHEDULE AN UPSTREAM.BREAK.DOWN GIVEN LOOP.BD.MSG NOW
    SCHEDULE A FINAL.ASSIGNMENT.NOTICE GIVEN CONT.MSG NOW
    GO TO RETN
  ALWAYS
LOOP
''
''   NEXT CHECK TO SEE IF THERE IS A SLOT AVAILABLE AT THE CALLED.NODE
''     TO ACCOMODATE THE RETURN TRANSMISSION OF A SLOT ASSIGNMENT AND
''     RECIPROCAL REQUEST FOR SERVICE.  NOTE:  WHENEVER A NODE IS NOT
''     TRANSMITTING, IT IS "LISTENING" TO ITS NEIGHBORS AND THEREFORE
''     KNOWS WHEN A NEIGHBOR MAY RECEIVE.
''
FOR J = 1 TO SLOTS, DO
  IF USE(CALLED.NODE,J,1) EQ 0 AND USE(CALLED.NODE,J,4) EQ 0 AND
    USE(CALLING.NODE,J,1) EQ 0
    GO TO NEXT1
  ALWAYS
LOOP
IF PRNT LE 4
  PRINT 4 LINES WITH CKT.NR(MESSAGE), ORIGINATOR(MESSAGE),
    DESTINATION(MESSAGE), TIME.V, TO.NODE(MESSAGE), FM.NODE(MESSAGE) AND
    (HOP.COUNT(MESSAGE) + 0.5) AS FOLLOWS
CIRCUIT NR. **** FROM NODE **  TO NODE ** COULD NOT BE ESTABLISHED AT
  THIS TIME, TIME.V = ****.******, BECAUSE THERE WERE NO MUTUALLY AVAIL-
  ABLE SLOTS BETWEEN NODES ** AND ** ON HOP **.* FOR USE IN TRANSMITTING
  THE RETURN SLOT ASSIGNMENT TO THE CALLING.NODE.
  SKIP 1 OUTPUT LINE
ALWAYS
LET CKT.FAILED = CKT.FAILED + 1
LET UP.ROUTE = UP.ROUTE - 1
''
''   CHECK IF THIS IS THE FIRST HOP OF THE MESSAGE.  IF IT IS, THEN THE
''     BREAK DOWN CIRCUIT ROUTINE NEED NOT BE CALLED SINCE NO SLOTS HAVE
''     BEEN ASSIGNED YET.  IF NOT, CALL THE BREAK DOWN CIRCUIT ROUTINE.
''
IF ORIGINATOR(MESSAGE) EQ FM.NODE(MESSAGE)
  LET P.BD.COUNTER = P.BD.COUNTER + 1
  DESTROY THE MESSAGE CALLED SVC1.MSG
  GO TO RETN
ALWAYS
''
'EXIT'
LET DOWN.ROUTE = DOWN.ROUTE + 1
LET TYPE(MESSAGE) = PARTIAL.BREAKDOWN
LET DIRECTION(MESSAGE) = 3
''
''   SINCE NO SLOTS ARE AVAILABLE TO CARRY A SLOT ASSIGNMENT OR BREAK
```

Drop

```
''      DOWN NOTICE BACK TO THE CALLING.NODE, SCHEDULE THE BREAK DOWN
''      TO COMMENCE AUTOMATICALLY AT THE CALLING.NODE AFTER A DELAY OF
''      (SLOTS + 2) * SLOT.DURATION UNITS TO SIMULATE THE REQUIREMENT
''      "TIMING OUT".  IF A SLOT WERE AVAILABLE, THE RESPONSE.REQ.FOR.SVC
''      WOULD BE RECEIVED BACK AT THE CALLING.NODE BEFORE THE SIGNAL
''      REQUIREMENT "TIMED OUT" OR EXPIRED.
''
LET START.TIME(MESSAGE) = TIME.V
IF SLOT.ARRIVAL(MESSAGE) LE SLOTS - 2
  LET SLOT.ARRIVAL(MESSAGE) = SLOT.ARRIVAL(MESSAGE) + 2
ALWAYS
IF SLOT.ARRIVAL(MESSAGE) EQ SLOTS - 1
  LET SLOT.ARRIVAL(MESSAGE) = 1
ALWAYS
IF SLOT.ARRIVAL(MESSAGE) EQ SLOTS
  LET SLOT.ARRIVAL(MESSAGE) = 2
ALWAYS
SCHEDULE A DOWNSTREAM.BREAK.DOWN GIVEN MESSAGE IN (REAL.F(SLOTS + 2) *
  SLOT.DURATION) UNITS
GO TO RETN
''
''    NOW WE CAN FIND THE NEXT MUTUALLY AVAILABLE SLOT (AT LEAST 1 FULL
''      SLOT IN THE FUTURE TO ACCOUNT FOR PROCESSING TIME IN THIS THE
''      CALLED.NODE) TO CARRY THE SLOT ASSIGNMENT AND RECIPROCAL REQUEST
''      FOR SERVICE BACK TO THE CALLING.NODE.
''
'NEXT1'
LET SLOT2 = 0
LET FRAME2 = 0
LET CURRENT.SLOT = SLOT.ARRIVAL(MESSAGE)
IF CURRENT.SLOT EQ (SLOTS - 1)
  LET L = 1
  GO TO SEARCH.NEXT.FRAME
ALWAYS
IF CURRENT.SLOT EQ SLOTS
  LET L = 2
  GO TO SEARCH.NEXT.FRAME
ALWAYS
LET L = CURRENT.SLOT + 2
FOR J = L TO SLOTS, DO
  IF USE(CALLED.NODE,J,1) EQ 0 AND USE(CALLED.NODE,J,4) EQ 0 AND
     USE(CALLING.NODE,J,1) EQ 0
     LET SLOT2 = J
     GO TO NEXT2
  ALWAYS
LOOP
LET L = 1
''
'SEARCH.NEXT.FRAME'
LET FRAME2 = 1
FOR J = L TO SLOTS, DO
  IF USE(CALLED.NODE,J,1) EQ 0 AND USE(CALLED.NODE,J,4) EQ 0 AND
     USE(CALLING.NODE,J,1) EQ 0
     LET SLOT2 = J
     GO TO NEXT2
  ALWAYS
LOOP
IF USE(CALLED.NODE,1,1) EQ 0 AND USE(CALLED.NODE,1,4) EQ 0 AND
  USE(CALLING.NODE,1,1) EQ 0
  LET FRAME2 = 2
  LET SLOT2 = 1
  GO TO NEXT2
ALWAYS
PRINT 1 LINE WITH CKT.NR(MESSAGE) AS FOLLOWS
SECONDARY SVC MSG FOR CKT ***** IN ERROR IN EVENT INITIAL.REQ.FOR.SVC
SKIP 1 OUTPUT LINE
LET CKT.FAILED = CKT.FAILED + 1
LET UP.ROUTE = UP.ROUTE - 1
IF ORIGINATOR(MESSAGE) EQ FM.NODE(MESSAGE)
  LET P.BD.COUNTER = P.BD.COUNTER + 1
  DESTROY THE MESSAGE CALLED SVC1.MSG
  GO TO RETN
```

```
ALWAYS
GO TO EXIT
''
''    IF WE GET AS FAR AS NEXT2 THEN WE HAVE IDENTIFIED THE SLOT TO CARRY
''      THE SLOT ASSIGNMENT AND RECIPROCAL REQUEST BACK TO THE CALL-
''      ING.NODE.  NOW CALCULATE WHEN THE SERVICE MESSAGE WILL ARRIVE AT
''      THE CALLING.NODE.  SCHEDULE ITS ARRIVAL AFTER THE SLOT ASSIGNMENT
''      IS MADE LATER IN THIS EVENT.
''
'NEXT2'
IF FRAME2 EQ 0
   LET DELAY2 = (REAL.F(SLOT2 - CURRENT.SLOT)) * SLOT.DURATION
   GO TO NEXT3
ALWAYS
IF FRAME2 EQ 1
   LET X = ((SLOTS + 1) - CURRENT.SLOT)
   LET Y = SLOT2 - 1
   LET DELAY2 = (REAL.F(X + Y)) * SLOT.DURATION
   GO TO NEXT3
ALWAYS
IF FRAME2 EQ 2
   LET DELAY2 = (REAL.F(SLOTS + 1)) * SLOT.DURATION
   GO TO NEXT3
ALWAYS
PRINT 1 LINE WITH CKT.NR(MESSAGE) AS FOLLOWS
ERROR IN CALCULATING DELAY2 IN EVENT  INITIAL.REQ.FOR.SVC, CKT.NR = *****
SKIP 1 OUTPUT LINE
LET CKT.FAILED = CKT.FAILED + 1
LET UP.ROUTE = UP.ROUTE - 1
IF ORIGINATOR(MESSAGE) EQ FM.NODE(MESSAGE)
   LET P.BD.COUNTER = P.BD.COUNTER + 1
   DESTROY THE MESSAGE CALLED SVC1.MSG
   GO TO RETN
ALWAYS
GO TO EXIT
''
''    IF WE GET AS FAR AS NEXT3 THEN WE HAVE IDENTIFIED A SLOT TO CARRY
''      THE SERVICE MESSAGE AND HAVE CALCULATED THE DELAY REQUIRED TO
''      SCHEDULE THE ARRIVAL OF THIS SERVICE MESSAGE BACK AT THE CALL-
''      ING.NODE.
''
''    NOW MAKE THE ACTUAL SLOT ASSIGNMENT FOR THE CALLING.NODE TO USE TO
''      TRANSMIT TO THE CALLED.NODE.  THE CALLED.NODE APPLIES THE SLOT
''      SELECTION ALGORITHM TO SELECT A SLOT WHICH STACKS THE RECEIVE
''      SIGNALS TO SOME "MAX.SLOT.DEPTH".
''
'NEXT3'
FOR I BACK FROM MAX.SLOT.DEPTH TO 2, DO
   FOR J = 1 TO SLOTS, DO
      IF USE(CALLED.NODE,J,1) EQ 0 AND USE(CALLED.NODE,J,4) EQ (I - 1)
         AND USE(CALLING.NODE,J,1) EQ 0 AND USE(CALLING.NODE,J,4) EQ 0
         LET SLOT.REC = J
         GO TO NEXT4
      ALWAYS
   LOOP
LOOP
''
''    IF PROCESSING PASSES THROUGH THE NESTED DO LOOPS ABOVE THEN WE CAN-
''      NOT STACK THE RECEIVE SIGNAL AND MUST EXAMINE THE POSSIBILITY OF
''      ASSIGNING AN EMPTY SLOT AS THE NEW RECEIVE SLOT.  THIS SLOT MUST
''      BE AT LEAST 1 FULL SLOT IN THE FUTURE TO ACCOUNT FOR PROCESSING
''      TIME IN THIS THE CALLED.NODE.
''
IF CURRENT.SLOT EQ (SLOTS - 1)
   LET M = 1
   GO TO FIND.RECEIVE.SLOT.IN.NEXT.FRAME
ALWAYS
IF CURRENT.SLOT EQ SLOTS
   LET M = 2
   GO TO FIND.RECEIVE.SLOT.IN.NEXT.FRAME
ALWAYS
LET M = CURRENT.SLOT + 2
```

```
FOR I = M TO SLOTS, DO
  IF USE(CALLED.NODE,I,1) EQ 0 AND USE(CALLED.NODE,I,4) EQ 0 AND
     USE(CALLING.NODE,I,1) EQ 0 AND USE(CALLING.NODE,I,4) EQ 0
     LET SLOT.REC = I
     GO TO NEXT4
  ELSE
LOOP
LET M = 1
''
'FIND.RECEIVE.SLOT.IN.NEXT.FRAME'
LET FRAME.REC = 1
FOR J = M TO SLOTS, DO
  IF USE(CALLED.NODE,J,1) EQ 0 AND USE(CALLED.NODE,J,4) EQ 0 AND
     USE(CALLING.NODE,J,1) EQ 0 AND USE(CALLING.NODE,J,4) EQ 0
     LET SLOT.REC = J
     GO TO NEXT4
  ALWAYS
LOOP
IF USE(CALLED.NODE,1,1) EQ 0 AND USE(CALLED.NODE,1,4) EQ 0 AND
  USE(CALLING.NODE,1,1) EQ 0 AND USE(CALLING.NODE,1,4) EQ 0
  LET FRAME.REC = 2
  LET SLOT.REC = 1
  GO TO NEXT4
ALWAYS
IF PRNT LE 4
  PRINT 4 LINES WITH CKT.NR(MESSAGE), ORIGINATOR(MESSAGE), DESTINATION
     (MESSAGE), TIME.V, HOP.COUNT(MESSAGE), TO.NODE(MESSAGE) AND FM.NODE
     (MESSAGE) AS FOLLOWS
CIRCUIT.NR. ***** FROM NODE ** TO NODE ** COMMENCING BREAKDOWN AT
  TIME.V = ****.****** AFTER *.** HOPS WERE COMPLETED BECAUSE THE
  CALLED.NODE (NODE **) RECOGNIZED THAT THERE WERE NO MUTUALLY AVAILABLE
  SLOTS FOR ASSIGNMENT BETWEEN THIS NODE AND THE CALLING.NODE (NODE **).
  SKIP 1 OUTPUT LINE
ALWAYS
LET CKT.FAILED = CKT.FAILED + 1
LET UP.ROUTE = UP.ROUTE - 1
IF ORIGINATOR(MESSAGE) EQ FM.NODE(MESSAGE)
  LET P.BD.COUNTER = P.BD.COUNTER + 1
  DESTROY THE MESSAGE CALLED SVC1.MSG
  GO TO RETN
ALWAYS
GO TO EXIT
''
''   WE CAN NOW MAKE THE SLOT ASSIGNMENT, UPDATE THE MESSAGE, AND
''      SCHEDULE THE "RESPONSE.REQ.FOR.SVC" AT THE CALLING.NODE
''
'NEXT4'
LET USE(CALLED.NODE,SLOT.REC,4) = USE(CALLED.NODE,SLOT.REC,4) + 1
LET CHANGE.FLAG = 1
LET SLOT.ARRIVAL(MESSAGE) = SLOT2
LET SLOT.ASSIGN(MESSAGE) = SLOT.REC
LET RECSLOT(MESSAGE) = SLOT.REC
SCHEDULE A RESPONSE.REQ.FOR.SVC GIVEN MESSAGE IN DELAY2 UNITS
IF PRNT LE 1
  PRINT 2 LINES WITH CKT.NR(MESSAGE), FM.NODE(MESSAGE), (TIME.V +
     DELAY2) AND DELAY2 AS FOLLOWS
  CIRCUIT NR. ***** HAS SCHEDULED A RESPONSE.REQ.FOR.SVC AT NODE ** AT
     TIME.V = ****.******, I.E. *.****** SECONDS FROM NOW.
  SKIP 1 OUTPUT LINE
  PRINT 1 LINE AS FOLLOWS
  ATTRIBUTES OF MESSAGE ENTITY AT END OF INITIAL.REQ.FOR.SVC ARE:
  LIST ATTRIBUTES OF MESSAGE
  SKIP 1 OUTPUT LINE
ALWAYS
''
'RETN'
RETURN
END ''OF INITIAL.REQ.FOR.SVC
''
''   THIS EVENT SIMULATES THE RESPONSE REQUEST FOR SERVICE FROM A
''      CALLED.NODE BACK TO A CALLING.NODE.  THE PROCESSING SIMULATED
''      HEREIN IS DONE AT THE CALLING.NODE.
```

```
''
EVENT RESPONSE.REQ.FOR.SVC GIVEN SVC2.MSG
LET MESSAGE = SVC2.MSG
IF PRNT LE 1
  PRINT 2 LINES WITH TIME.V AS FOLLOWS
EVENT RESPONSE.REQ.FOR.SVC INVOKED AT TIME.V = ****.******
------------------------------------------------------------
  SKIP 1 OUTPUT LINE
ALWAYS
IF PRNT LE 3
  PRINT 1 LINE AS FOLLOWS
  ATTRIBUTES OF MESSAGE ENTITY AT THE START OF RESPONSE.REQ.FOR.SVC ARE:
  LIST ATTRIBUTES OF MESSAGE
  SKIP 1 OUTPUT LINE
ALWAYS
DEFINE DELAY3 AS A REAL VARIABLE
LET FRAME.REC = 0
LET SLOT.REC = 0
''
''   FIRST CHECK TO SEE IF THERE IS STILL A SLOT AVAILABLE AT THE CALL-
''     ING.NODE TO ACCOMODATE THE RETURN TRANSMISSION OF A SLOT ASSIGN-
''     MENT TO THE CALLED.NODE.  NOTE:  AS ALWAYS, WHENEVER A NODE IS
''     NOT TRANSMITTING, IT IS "LISTENING" TO ITS NEIGHBORS AND THERE-
''     FORE KNOWS IF AND WHEN A NEIGHBOR MAY RECEIVE.
''
LET CALLING.NODE = FM.NODE(MESSAGE)
LET CALLED.NODE = TO.NODE(MESSAGE)
FOR J = 1 TO SLOTS, DO
  IF USE(CALLING.NODE,J,1) EQ 0 AND USE(CALLING.NODE,J,4) EQ 0 AND
    USE(CALLED.NODE,J,1) EQ 0
    GO TO PASS1
  ALWAYS
LOOP
IF PRNT LE 1
  PRINT 4 LINES WITH CKT.NR(MESSAGE), ORIGINATOR(MESSAGE),
    DESTINATION(MESSAGE), TIME.V, FM.NODE(MESSAGE), TO.NODE(MESSAGE) AND
    (HOP.COUNT(MESSAGE) + 0.5) AS FOLLOWS
CIRCUIT NR. ***** FROM NODE ** TO NODE ** COULD NOT BE ESTABLISHED AT
  THIS TIME, TIME.V = ****.******, BECAUSE THERE WERE NO MUTUALLY AVAIL-
  ABLE SLOTS BETWEEN NODES ** AND ** ON HOP **.* FOR USE IN TRANSMITTING
  THE RETURN SLOT ASSIGNMENT TO THE CALLED.NODE.
  SKIP 1 OUTPUT LINE
ALWAYS
''
'XSIT'
''
''   SINCE NO SLOTS ARE AVAILABLE TO CARRY A SLOT ASSIGNMENT OR BREAK
''     DOWN NOTICE BACK TO THE CALLED.NODE, SCHEDULE THE BREAK DOWN TO
''     COMMENCE AUTOMATICALLY AT THE CALLED.NODE AFTER A DELAY OF (SLOTS
''     + 2) * SLOT.DURATION UNITS TO SIMULATE THE REQUIREMENT "TIMING
''     OUT" OR EXPIRING.  IF THE SLOT THAT WAS JUST ASSIGNED BY THE CAL-
''     LED.NODE WERE STILL AVAILABLE AT THE CALLING.NODE, IT COULD BE
''     USED AND THE FINAL.ASSIGNMENT.NOTICE WOULD BE RECEIVED BACK AT
''     THE CALLED.NODE BEFORE THE SIGNAL REQUIREMENT "TIMED OUT".
''
PERFORM CKT.IS.NOT.ESTAB GIVEN MESSAGE
LET START.TIME(MESSAGE) = TIME.V
IF SLOT.ARRIVAL(MESSAGE) LE SLOTS - 2
  LET SLOT.ARRIVAL(MESSAGE) = SLOT.ARRIVAL(MESSAGE) + 2
ALWAYS
IF SLOT.ARRIVAL(MESSAGE) EQ SLOTS - 1
  LET SLOT.ARRIVAL(MESSAGE) = 1
ALWAYS
IF SLOT.ARRIVAL(MESSAGE) EQ SLOTS
  LET SLOT.ARRIVAL(MESSAGE) = 2
ALWAYS
IF FM.NODE(MESSAGE) EQ ORIGINATOR(MESSAGE)
  CREATE A MESSAGE CALLED NEW.MSG
    LET CKT.NR(NEW.MSG) = CKT.NR(MESSAGE)
    LET TYPE(NEW.MSG) = PARTIAL.BREAKDOWN
    LET ORIGINATOR(NEW.MSG) = ORIGINATOR(MESSAGE)
    LET DESTINATION(NEW.MSG) = DESTINATION(MESSAGE)
```

```
        LET FM.NODE(NEW.MSG) = FM.NODE(MESSAGE)
        LET TO.NODE(NEW.MSG) = TO.NODE(MESSAGE)
        LET START.TIME(NEW.MSG) = TIME.V
        LET HOP.COUNT(NEW.MSG) = HOP.COUNT(MESSAGE)
        LET SLOT.ARRIVAL(NEW.MSG) = SLOT.ARRIVAL(MESSAGE)
        LET SLOT.ASSIGN(NEW.MSG) = SLOT.ASSIGN(MESSAGE)
        LET RECSLOT(NEW.MSG) = SLOTS + 1
        LET DIRECTION(NEW.MSG) = 0
        LET CUM.ENERGY(NEW.MSG) = 0.0
        LET INFO1(NEW.MSG) = INFO1(MESSAGE)
        LET INFO2(NEW.MSG) = INFO2(MESSAGE)
        LET INFO3(NEW.MSG) = INFO3(MESSAGE)
        LET INFO4(NEW.MSG) = INFO4(MESSAGE)
        LET INFO5(NEW.MSG) = INFO5(MESSAGE)
        LET INFO6(NEW.MSG) = INFO6(MESSAGE)
        LET INFO7(NEW.MSG) = INFO7(MESSAGE)
        LET INFO8(NEW.MSG) = INFO8(MESSAGE)
        LET INFO9(NEW.MSG) = INFO9(MESSAGE)
     DESTROY THE MESSAGE CALLED SVC2.MSG
     SCHEDULE AN UPSTREAM.BREAK.DOWN GIVEN NEW.MSG IN (REAL.F(SLOTS + 2) *
        SLOT.DURATION) UNITS
     GO TO RETRN
ALWAYS
CREATE A MESSAGE CALLED NEW.MSG
     LET CKT.NR(NEW.MSG) = CKT.NR(MESSAGE)
     LET TYPE(NEW.MSG) = PARTIAL.BREAKDOWN
     LET ORIGINATOR(NEW.MSG) = ORIGINATOR(MESSAGE)
     LET DESTINATION(NEW.MSG) = DESTINATION(MESSAGE)
     LET FM.NODE(NEW.MSG) = FM.NODE(MESSAGE)
     LET TO.NODE(NEW.MSG) = TO.NODE(MESSAGE)
     LET START.TIME(NEW.MSG) = TIME.V
     LET SLOT.ARRIVAL(NEW.MSG) = SLOT.ARRIVAL(MESSAGE)
     LET SLOT.ASSIGN(NEW.MSG) = SLOT.ASSIGN(MESSAGE)
     LET RECSLOT(NEW.MSG) = RECSLOT(MESSAGE)
     LET DIRECTION(NEW.MSG) = 0
     LET CUM.ENERGY(NEW.MSG) = 0.0
     LET INFO1(NEW.MSG) = INFO1(MESSAGE)
     LET INFO2(NEW.MSG) = INFO2(MESSAGE)
     LET INFO3(NEW.MSG) = INFO3(MESSAGE)
     LET INFO4(NEW.MSG) = INFO4(MESSAGE)
     LET INFO5(NEW.MSG) = INFO5(MESSAGE)
     LET INFO6(NEW.MSG) = INFO6(MESSAGE)
     LET INFO7(NEW.MSG) = INFO7(MESSAGE)
     LET INFO8(NEW.MSG) = INFO8(MESSAGE)
     LET INFO9(NEW.MSG) = INFO9(MESSAGE)
SCHEDULE AN UPSTREAM.BREAK.DOWN GIVEN NEW.MSG IN (REAL.F(SLOTS + 2) *
     SLOT.DURATION) UNITS
LET DIRECTION(MESSAGE) = 4
SCHEDULE A DOWNSTREAM.BREAK.DOWN GIVEN MESSAGE NOW
GO TO RETRN
''
''
''     NOW WE CAN FIND THE NEXT MUTUALLY AVAILABLE SLOT (AT LEAST 1 FULL
''        SLOT IN THE FUTURE TO ACCOUNT FOR PROCESSING TIME IN THIS THE
''        CALLING.NODE) TO CARRY THE SLOT ASSIGNMENT BACK TO THE
''        CALLED.NODE.
''
'PASS1'
LET SLOT3 = 0
LET FRAME3 = 0
LET CURRENT.SLOT = SLOT.ARRIVAL(MESSAGE)
IF CURRENT.SLOT EQ (SLOTS - 1)
     LET N = 1
     GO TO SEARCH.NEXT.FRAME
ALWAYS
IF CURRENT.SLOT EQ SLOTS
     LET N = 2
     GO TO SEARCH.NEXT.FRAME
ALWAYS
LET N = CURRENT.SLOT + 2
FOR J = N TO SLOTS, DO
     IF USE(CALLING.NODE,J,1) EQ 0 AND USE(CALLING.NODE,J,4) EQ 0 AND
```

```
      USE(CALLED.NODE,J,1) EQ 0
      LET SLOT3 = J
      GO TO PASS2
   ALWAYS
LOOP
LET N = 1
''
'SEARCH.NEXT.FRAME'
LET FRAME3 = 1
FOR J = N TO SLOTS, DO
   IF USE(CALLING.NODE,J,1) EQ 0 AND USE(CALLING.NODE,J,4) EQ 0 AND
      USE(CALLED.NODE,J,1) EQ 0
      LET SLOT3 = J
      GO TO PASS2
   ALWAYS
LOOP
IF USE(CALLING.NODE,1,1) EQ 0 AND USE(CALLING.NODE,1,4) EQ 0 AND
   USE(CALLED.NODE,1,1) EQ 0
   LET FRAME3 = 2
   LET SLOT3 = 1
   GO TO PASS2
ALWAYS
PRINT 1 LINE WITH CKT.NR(MESSAGE) AS FOLLOWS
TERTIARY SVC MSG FOR CKT ***** IN ERROR IN EVENT RESPONSE.REQ.FOR.SVC
SKIP 1 OUTPUT LINE
GO TO XSIT
''
''    IF WE GET AS FAR AS PASS2 THEN WE HAVE IDENTIFIED THE SLOT TO CARRY
''       THE SLOT ASSIGNMENT BACK TO THE CALLED.NODE.  NOW CALCULATE WHEN
''       THE SERVICE MESSAGE WILL ARRIVE AT THE CALLED.NODE.  WE SHALL
''       SCHEDULE ITS ARRIVAL AFTER THE SLOT ASSIGNMENT HAS BEEN MADE
''       LATER IN THIS EVENT.
''
'PASS2'
IF FRAME3 = 0
   LET DELAY3 = (REAL.F(SLOT3 - CURRENT.SLOT)) * SLOT.DURATION
   GO TO PASS3
ALWAYS
IF FRAME3 EQ 1
   LET X = ((SLOTS + 1) - CURRENT.SLOT)
   LET Y = SLOT3 - 1
   LET DELAY3 = (REAL.F(X + Y)) * SLOT.DURATION
   GO TO PASS3
ALWAYS
IF FRAME3 EQ 2
   LET DELAY3 = (REAL.F(SLOTS + 1)) * SLOT.DURATION
   GO TO PASS3
ALWAYS
PRINT 1 LINE WITH CKT.NR(MESSAGE) AS FOLLOWS
ERROR IN CALCULATING DELAY3 IN EVENT RESPONSE.REQ.FOR.SVC, CKT.NR =*****
SKIP 1 OUTPUT LINE
GO TO XSIT
''
''    IF WE GET AS FAR AS PASS3 THEN WE HAVE IDENTIFIED A SLOT TO CARRY
''       THE SERVICE MESSAGE SLOT ASSIGNMENT AND HAVE CALCULATED THE DELAY
''       REQUIRED TO SCHEDULE THE ARRIVAL OF THIS SERVICE MESSAGE BACK AT
''       THE CALLED.NODE.
''
''    NOW APPLY THE SLOT SELECTION ALGORITHM TO SELECT A SLOT WHICH PER-
''       MITS RECEIVE SIGNALS TO BE STACKED TO SOME "MAX.SLOT.DEPTH".  THE
''       ACTUAL SLOT ASSIGNMENT IS MADE BY THE CALLING.NODE.
''
'PASS3'
FOR I BACK FROM MAX.SLOT.DEPTH TO 2, DO
   FOR J = 1 TO SLOTS, DO
      IF USE(CALLING.NODE,J,1) EQ 0 AND USE(CALLING.NODE,J,4) EQ (I - 1)
         AND USE(CALLED.NODE,J,1) EQ 0 AND USE(CALLED.NODE,J,4) EQ 0
         LET SLOT.REC = J
         GO TO PASS4
      ALWAYS
   LOOP
LOOP
```

```
''
''   IF PROCESSING PASSES THROUGH THE NESTED DO LOOPS ABOVE THEN WE CAN-
''     NOT STACK THE RECEIVE SIGNAL AND MUST EXAMINE THE POSSIBILITY OF
''     ASSIGNING AN EMPTY SLOT AS THE NEW RECEIVE SLOT.  THIS SLOT MUST
''     BE AT LEAST 1 FULL SLOT IN THE FUTURE TO ACCOUNT FOR PROCESSING
''     TIME IN THIS THE CALLING.NODE.
''
IF CURRENT.SLOT EQ (SLOTS - 1)
  LET M = 1
  GO TO FIND.RECEIVE.SLOT.IN.NEXT.FRAME
ALWAYS
IF CURRENT.SLOT EQ SLOTS
  LET M = 2
  GO TO FIND.RECEIVE.SLOT.IN.NEXT.FRAME
ALWAYS
LET M = CURRENT.SLOT + 2
FOR I = M TO SLOTS, DO
  IF USE(CALLING.NODE,I,1) EQ 0 AND USE(CALLING.NODE,I,4) EQ 0 AND
    USE(CALLED.NODE,I,1) EQ 0 AND USE(CALLED.NODE,I,4) EQ 0
    LET SLOT.REC = I
    GO TO PASS4
  ELSE
LOOP
LET M = 1
''
'FIND.RECEIVE.SLOT.IN.NEXT.FRAME'
LET FRAME.REC = 1
FOR J = M TO SLOTS, DO
  IF USE(CALLING.NODE,J,1) EQ 0 AND USE(CALLING.NODE,J,4) EQ 0 AND
    USE(CALLED.NODE,J,1) EQ 0 AND USE(CALLED.NODE,J,4) EQ 0
    LET SLOT.REC = J
    GO TO PASS4
  ALWAYS
LOOP
IF USE(CALLING.NODE,1,1) EQ 0 AND USE(CALLING.NODE,1,4) EQ 0 AND
  USE(CALLED.NODE,1,1) EQ 0 AND USE(CALLED.NODE,1,4) EQ 0
  LET FRAME.REC = 2
  LET SLOT.REC = 1
  GO TO PASS4
ALWAYS
IF PRNT LE 1
  PRINT 4 LINES WITH CKT.NR(MESSAGE), ORIGINATOR(MESSAGE), DESTINATION
    (MESSAGE), TIME.V, HOP.COUNT(MESSAGE), FM.NODE(MESSAGE) AND TO.NODE
    (MESSAGE) AS FOLLOWS
CIRCUIT NR. ***** FROM NODE ** TO NODE ** COMMENCING BREAK DOWN AS
  TIME.V = ****.****** AFTER *.** HOPS WERE COMPLETED BECAUSE THE CALL-
  ING.NODE (NODE **) RECOGNIZED THAT THERE WERE NO MUTUALLY AVAILABLE
  SLOTS FOR ASSIGNMENT BETWEEN THIS NODE AND THE CALLED.NODE (NODE **).
  SKIP 1 OUTPUT LINE
ALWAYS
GO TO XSIT
''
''   CHECK TO SEE IF THE SLOT THE CALLED.NODE WANTS TO ASSIGN AS OUR
''     TRANSMIT SLOT IS STILL AVAILABLE.  IT MAY HAVE BEEN ASSIGNED FOR
''     SOME OTHER USE DURING THE LAST SEVERAL MILLISECONDS WHILE THE 2
''     NODES WERE COORDINATING.
''
'PASS4'
IF USE(CALLING.NODE,SLOT.ASSIGN(MESSAGE),1) NE 0 OR USE(CALLING.NODE,
  SLOT.ASSIGN(MESSAGE),4) NE 0
  IF LTD.PRINT EQ 0
  PRINT 3 LINES WITH CKT.NR(MESSAGE) AS FOLLOWS
CIRCUIT NR. ***** IS EITHER EXPERIENCING AN ERROR OR HAS HAD ITS ANTICI-
  PATED UPSTREAM TRANSMIT SLOT ASSIGNED FOR SOME OTHER PURPOSE A SPLIT-
  SECOND BEFORE THIS ASSIGNMENT WAS RECEIVED IN RESPONSE.REQ.FOR.SVC.
  SKIP 1 OUTPUT LINE
  ALWAYS
  GO TO XSIT
ALWAYS
''
LET HOP.COUNT(MESSAGE) = HOP.COUNT(MESSAGE) + 0.5
''
```

```
''   WE CAN NOW MAKE THE SLOT ASSIGNMENT, UPDATE THE MESSAGE, AND
''     SCHEDULE THE "FINAL.ASSIGNMENT.NOTICE" AT THE CALLED.NODE.
''
LET USE(CALLING.NODE,SLOT.REC,4) = USE(CALLING.NODE,SLOT.REC,4) + 1
LET USE(CALLING.NODE,SLOT.ASSIGN(MESSAGE),1) = CKT.NR(MESSAGE)
LET USE(CALLING.NODE,SLOT.ASSIGN(MESSAGE),2) = SLOT.REC
LET USE(CALLING.NODE,SLOT.ASSIGN(MESSAGE),3) = CALLED.NODE
LET CHANGE.FLAG = 1
''
LET SLOT.ARRIVAL(MESSAGE) = SLOT3
LET SLOT.ASSIGN(MESSAGE) = SLOT.REC
SCHEDULE A FINAL.ASSIGNMENT.NOTICE GIVEN MESSAGE IN DELAY3 UNITS
IF PRNT LE 1
   PRINT 2 LINES WITH CKT.NR(MESSAGE), TO.NODE(MESSAGE), (TIME.V +
     DELAY3) AND DELAY3 AS FOLLOWS
   CIRCUIT NR. ***** HAS SCHEDULED A FINAL.ASSIGNMENT.NOTICE AT NODE **
     AT TIME.V = ****.******, I.E. *.****** SECONDS FROM NOW.
   SKIP 1 OUTPUT LINE
   PRINT 1 LINE AS FOLLOWS
   ATTRIBUTES OF MESSAGE ENTITY AT THE END OF RESPONSE.REQ.FOR.SVC ARE:
   LIST ATTRIBUTES OF MESSAGE
   SKIP 1 OUTPUT LINE
ALWAYS
''
'RETRN'
RETURN
END ''OF RESPONSE.REQ.FOR.SVC
''
''   THIS EVENT SIMULATES THE ACTIONS PERFORMED AT THE CALLED.NODE WHEN
''     THE FINAL SERVICE OR COORDINATION SLOT ASSIGNMENT MESSAGE IS
''     RECEIVED FROM THE CALLING.NODE.  THE CALLED.NODE MAY BE THE
''     DESTINATION NODE FOR THE CIRCUIT OR MIGHT ONLY BE ONE OF THE
''     INTERMEDIATE NODES ON THE BEST.PATH TO THE DESTINATION.
''
EVENT FINAL.ASSIGNMENT.NOTICE GIVEN SVC3.MSG
LET MESSAGE = SVC3.MSG
DEFINE DELAY4 AS A REAL VARIABLE
''
''   FIRST TEST TO SEE IF THIS IS THE CONTINUATION OF A SHORT CIRCUIT
''     LOOP MESSAGE.
''
IF SLOT.ASSIGN(MESSAGE) EQ SLOTS + 1
  GO TO CONT1
ALWAYS
''
IF PRNT LE 1
   PRINT 2 LINES WITH TIME.V AS FOLLOWS
EVENT FINAL.ASSIGNMENT.NOTICE INVOKED AT TIME.V = ****.******
-------------------------------------------------------------
   SKIP 1 OUTPUT LINE
ALWAYS
IF PRNT LE 3
   PRINT 1 LINE AS FOLLOWS
   ATTRIBUTES OF THE MESSAGE ENTITY AT START OF FINAL.ASSIGNMENT.NOTICE:
   LIST ATTRIBUTES OF MESSAGE
   SKIP 1 OUTPUT LINE
ALWAYS
''
LET CALLING.NODE = FM.NODE(MESSAGE)
LET CALLED.NODE = TO.NODE(MESSAGE)
''
''   CHECK TO SEE IF THE SLOT THE CALLING.NODE WANTS TO ASSIGN AS THIS
''     CALLED.NODE TRANSMIT SLOT IS STILL AVAILABLE.  IT MAY HAVE BEEN
''     ASSIGNED FOR SOME OTHER USE WHILE THE 2 NODES WERE COORDINATING.
''
IF USE(CALLED.NODE,SLOT.ASSIGN(MESSAGE),1) NE 0 OR USE(CALLED.NODE,
   SLOT.ASSIGN(MESSAGE),4) NE 0
   IF LTD.PRINT EQ 0
   PRINT 3 LINES WITH CKT.NR(MESSAGE) AS FOLLOWS
CIRCUIT NR. ***** IS EITHER EXPERIENCING AN ERROR OR HAS HAD ITS ANTICI-
   PATED DOWNSTREAM TRANSMIT SLOT ASSIGNED FOR SOME OTHER USE A SPLIT-
   SECOND BEFORE THIS ASSIGNMENT WAS RECEIVED IN FINAL.ASSIGNMENT.NOTICE.
```

```
   SKIP 1 OUTPUT LINE
   ALWAYS
   PERFORM CKT.IS.NOT.ESTAB GIVEN MESSAGE
   LET USE(CALLED.NODE,RECSLOT(MESSAGE),4) = USE(CALLED.NODE,
     RECSLOT(MESSAGE),4) - 1
   LET START.TIME(MESSAGE) = TIME.V
   IF SLOT.ARRIVAL(MESSAGE) LE SLOTS - 2
     LET SLOT.ARRIVAL(MESSAGE) = SLOT.ARRIVAL(MESSAGE) + 2
   ALWAYS
   IF SLOT.ARRIVAL(MESSAGE) EQ SLOTS - 1
     LET SLOT.ARRIVAL(MESSAGE) = 1
   ALWAYS
   IF SLOT.ARRIVAL(MESSAGE) EQ SLOTS
     LET SLOT.ARRIVAL(MESSAGE) = 2
   ALWAYS
   LET DIRECTION(MESSAGE) = 5
   SCHEDULE A DOWNSTREAM.BREAK.DOWN GIVEN MESSAGE IN (REAL.F(SLOTS + 2) *
     SLOT.DURATION) UNITS
   GO TO RETURN
ALWAYS
''
LET HOP.COUNT(MESSAGE) = HOP.COUNT(MESSAGE) + 0.5
LET CUM.ENERGY(MESSAGE) = CUM.ENERGY(MESSAGE) + ENERGY(CALLING.NODE,
   CALLED.NODE)
''
''    RECORD THE TRANSMIT SLOT ASSIGNMENT.
''
LET USE(CALLED.NODE,SLOT.ASSIGN(MESSAGE),1) = CKT.NR(MESSAGE)
LET USE(CALLED.NODE,SLOT.ASSIGN(MESSAGE),2) = RECSLOT(MESSAGE)
LET USE(CALLED.NODE,SLOT.ASSIGN(MESSAGE),3) = CALLING.NODE
LET USE(CALLED.NODE,SLOT.ASSIGN(MESSAGE),5) = INT.F(HOP.COUNT(MESSAGE))
LET USE(CALLED.NODE,SLOT.ASSIGN(MESSAGE),6) = INT.F(CUM.ENERGY(MESSAGE))
LET CHANGE.FLAG = 1
''
''    THE FOLLOWING BLOCK OF STATEMENTS STORE THE LINK NUMBER OF THE LINK
''      JUST USED TO CARRY THE MOST RECENT HOP OF THE CIRCUIT.  THIS INFO
''      WILL BE COLLECTED IN THE COMPLETED.CKT ROUTINE TO PRODUCE A LINK
''      USAGE REPORT AT THE END OF THE SIMULATION.  NOTE:  AFTER SEVERAL
''      WEEKS OF EXPERIENCED WITH THIS PROGRAM WE HAVE YET TO SEE ANY
''      CIRCUITS BACKTRACK OR LOOPBACK SO THE CODE SIMILAR TO THIS TO
''      REMOVE THE EFFECTS OF DETECTED AND SHORT CIRCUITED LOOPS HAS BEEN
''      OMITTED.  NO CIRCUIT HAS EVER BEEN OBSERVED TO MAKE MORE THAN
''      A TOTAL OF 8 HOPS.
''
LET H = INT.F(HOP.COUNT(MESSAGE))
IF H EQ 1
   LET INFO1(MESSAGE) = LI.NK.NR(CALLING.NODE,CALLED.NODE)
   GO TO AL~~ABLE~~ COMPLETE
ALWAYS
IF H EQ 2
   LET INFO2(MESSAGE) = LI.NK.NR(CALLING.NODE,CALLED.NODE)
   GO TO ALABLE
ALWAYS
IF H EQ 3
   LET INFO3(MESSAGE) = LI.NK.NR(CALLING.NODE,CALLED.NODE)
   GO TO ALABLE
ALWAYS
IF H EQ 4
   LET INFO4(MESSAGE) = LI.NK.NR(CALLING.NODE,CALLED.NODE)
   GO TO ALABLE
ALWAYS
IF H EQ 5
   LET INFO5(MESSAGE) = LI.NK.NR(CALLING.NODE,CALLED.NODE)
   GO TO ALABLE
ALWAYS
IF H EQ 6
   LET INFO6(MESSAGE) = LI.NK.NR(CALLING.NODE,CALLED.NODE)
   GO TO ALABLE
ALWAYS
IF H EQ 7
   LET INFO7(MESSAGE) = LI.NK.NR(CALLING.NODE,CALLED.NODE)
   GO TO ALABLE
```

```
ALWAYS
IF H EQ 8
  LET INFO8(MESSAGE) = LI.NK.NR(CALLING.NODE,CALLED.NODE)
  GO TO ALABLE
ALWAYS
IF H EQ 9
  LET INFO9(MESSAGE) = LI.NK.NR(CALLING.NODE,CALLED.NODE)
  GO TO ALABLE
ALWAYS
''
'ALABLE'
''
''   NEXT CHECK TO SEE IF THIS CIRCUIT IS NOW COMPLETE.  IF IT IS, CALL
''      THE "COMPLETED.CKT" ROUTINE AND COLLECT APPROPRIATE STATISTICS.
''
IF TO.NODE(MESSAGE) EQ DESTINATION(MESSAGE)
  LET START.TIME(MESSAGE) = TIME.V - START.TIME(MESSAGE)
  PERFORM COMPLETED.CKT GIVEN MESSAGE
  GO TO RETIRN
ALWAYS
''
''   IF THE CIRCUIT HAS NOT BEEN ESTABLISHED ALL THE WAY TO THE DESTINA-
''      TION, THEN WE MUST TAKE ACTION TO ESTABLISH THE NEXT LINK TO THE
''      DESTINATION.
''
IF TO.NODE(MESSAGE) NE DESTINATION(MESSAGE)
  LET FM.NODE(MESSAGE) = TO.NODE(MESSAGE)
  LET TO.NODE(MESSAGE) = BEST.PATH(FM.NODE(MESSAGE),
    DESTINATION(MESSAGE))
  GO TO CONT1
ALWAYS
PRINT 1 LINE WITH CKT.NR(MESSAGE) AS FOLLOWS
ERROR IN EVENT FINAL.ASSIGNMENT.NOTICE FOR CIRCUIT NR. *****
SKIP 1 OUTPUT LINE
''
''   THE REMAINDER OF THIS EVENT SIMULATES ACTIONS PERFORMED AT AN
''      INTERMEDIATE NODE ALONG A BEST PATH ROUTE FROM AN ORIGINATOR TO
''      A DESTINATION.  THE "CALLED.NODE" HAS BECOME THE NEW "CALL-
''      ING NODE" AS WE NOW ATTEMPT TO ESTABLISH THE NEXT LINK OF THE
''      CIRCUIT.  THE CODE THAT FOLLOWS IS VERY SIMILAR TO THE LAST HALF
''      OF THE CODE IN THE "NEW.CKT.RECMT" EVENT BECAUSE THE ACTIONS THAT
''      MUST NOW BE PERFORMED ARE SIMILAR TO THOSE THAT ARE DONE WHEN WE
''      FIRST CREATE A CIRCUIT REQUIREMENT AND START BUILDING THE FIRST
''      LINK OF THE CIRCUIT.
''
'CONT1'
''
''   FIRST CHECK TO SEE IF THERE IS A SLOT AVAILABLE AT THIS NEWLY
''      DESIGNATED CALLING.NODE TO ACCOMMODATE THE TRANSMISSION OF A SER-
''      VICE MESSAGE TO THE NEWLY DESIGNATED CALLED.NODE.  SINCE WE ARE
''      ASSUMING THAT EACH NODE IS ALWAYS LISTENING TO ITS NEIGHBORS, THE
''      CALLING.NODE KNOWS WHEN ITS NEIGHBORS ARE NOT TRANSMITTING AND,
''      THEREFORE, ARE ABLE TO RECEIVE.  NOTE:  ALL NODES "LISTEN" WHEN-
''      EVER THEY ARE NOT TRANSMITTING.
''
LET CALLING.NODE = FM.NODE(MESSAGE)
LET CALLED.NODE = TO.NODE(MESSAGE)
FOR J = 1 TO SLOTS, DO
  IF USE(CALLING.NODE,J,1) EQ 0 AND USE(CALLING.NODE,J,4) EQ 0 AND
    USE(CALLED.NODE,J,1) EQ 0
    GO TO CONT2
  ALWAYS
LOOP
IF PRNT LE 1
  PRINT 3 LINES WITH CKT.NR(MESSAGE), ORIGINATOR(MESSAGE),
    DESTINATION(MESSAGE), CALLING.NODE, CALLED.NODE AND
    (HOP.COUNT(MESSAGE) + 0.5) AS FOLLOWS
CIRCUIT NR. ***** FROM NODE ** TO NODE ** CANNOT BE ESTABLISHED AT THIS
  TIME BECAUSE THERE ARE NO MUTUALLY AVAILABLE SLOTS BETWEEN NODES **
  AND **  ON HOP **.* TO CARRY THE INITIAL SERVICE MESSAGE.
  SKIP 1 OUTPUT LINE
ALWAYS
```

```
''
'EXSIT'
PERFORM CKT.IS.NOT.ESTAB GIVEN MESSAGE
LET DIRECTION(MESSAGE) = 3
LET START.TIME(MESSAGE) = TIME.V
SCHEDULE A DOWNSTREAM.BREAK.DOWN GIVEN MESSAGE NOW
GO TO RETIRN
''
''   WE KNOW THAT THE "CURRENT.SLOT" IS CONTAINED IN THE MESSAGE ATTRI-
''      BUTE CALLED "SLOT.ARRIVAL".
''
'CONT2'
LET CURRENT.SLOT = SLOT.ARRIVAL(MESSAGE)
''
''   FIND THE NEXT MUTUALLY AVAILABLE SLOT (AT LEAST 1 FULL SLOT IN THE
''      FUTURE TO ACCOUNT FOR PROCESSING TIME IN THE CALLING.NODE).
''
LET SLOT4 = 0
LET FRAME4 = 0
IF CURRENT.SLOT EQ (SLOTS - 1)
   LET K = 1
   GO TO CHECK.NEXT.FRAME
ALWAYS
IF CURRENT.SLOT EQ SLOTS
   LET K = 2
   GO TO CHECK.NEXT.FRAME
ALWAYS
LET K = CURRENT.SLOT + 2
FOR J = K TO SLOTS, DO
   IF USE(CALLING.NODE,J,1) EQ 0 AND USE(CALLING.NODE,J,4) EQ 0 AND
      USE(CALLED.NODE,J,1) EQ 0
      LET SLOT4 = J
      GO TO CONT3
   ALWAYS
LOOP
LET K = 1
''
'CHECK.NEXT.FRAME'
LET FRAME4 = 1
FOR J = K TO SLOTS, DO
   IF USE(CALLING.NODE,J,1) EQ 0 AND USE(CALLING.NODE,J,4) EQ 0 AND
      USE(CALLED.NODE,J,1) EQ 0
      LET SLOT4 = J
      GO TO CONT3
   ALWAYS
LOOP
IF USE(CALLING.NODE,1,1) EQ 0 AND USE(CALLING.NODE,1,4) EQ 0 AND
   USE(CALLED.NODE,1,1) EQ 0
   LET FRAME4 = 2
   LET SLOT4 = 1
   GO TO CONT3
ALWAYS
PRINT 2 LINES WITH CKT.NR(MESSAGE) AND TIME.V AS FOLLOWS
SVC MSG SLOT ASSIGNMENT ERROR. EVENT FINAL.ASSIGN.NOTICE. CKT.NR = *****
   AND TIME.V = ****.******
SKIP 1 OUTPUT LINE
GO TO EXSIT
''
''   IF WE GET AS FAR AS CONT3 THEN WE HAVE IDENTIFIED A SLOT TO CARRY
''      THE SERVICE MESSAGE TO THE CALLED.NODE.  NOW CALCULATE WHEN THE
''      SERVICE MESSAGE WILL ARRIVE AT THE CALLED.NODE AND SCHEDULE ITS
''      ARRIVAL APPROPRIATELY.
''
'CONT3'
IF FRAME4 EQ 0
   LET DELAY4 = (REAL.F(SLOT4 - CURRENT.SLOT)) * SLOT.DURATION
   GO TO CONT4
ALWAYS
IF FRAME4 EQ 1
   LET X = ((SLOTS + 1) - CURRENT.SLOT)
   LET Y = SLOT4 - 1
   LET DELAY4 = (REAL.F(X + Y)) * SLOT.DURATION
```

```
  GO TO CONT4
ALWAYS
IF FRAME4 EQ 2
  LET DELAY4 = (REAL.F(SLOTS + 1)) * SLOT.DURATION
  GO TO CONT4
ALWAYS
PRINT 1 LINE WITH CKT.NR(MESSAGE) AS FOLLOWS
ERROR IN CALCULATING DELAY4 IN EVENT FINAL.ASSIGN.NOTICE. CKT.NR = *****
SKIP 1 OUTPUT LINE
GO TO EXSIT
''
'CONT4'
LET SLOT.ARRIVAL(MESSAGE) = SLOT4
LET SLOT.ASSIGN(MESSAGE) = 0
LET RECSLOT(MESSAGE) = 0
SCHEDULE AN INITIAL.REQ.FOR.SVC GIVEN MESSAGE IN DELAY4 UNITS
IF PRNT LE 1
  PRINT 3 LINES WITH CKT.NR(MESSAGE), HOP.COUNT(MESSAGE), CALLED.NODE,
    (TIME.V + DELAY4) AND DELAY4 AS FOLLOWS
CIRCUIT NR. ***** HAS COMPLETED **.* HOPS AND HAS SCHEDULED AN INITIAL.
  REQ.FOR.SVC AT NODE ** AT TIME.V = ****.****** SECONDS, I.E. *.******
  SECONDS FROM NOW.
  SKIP 1 OUTPUT LINE
  PRINT 1 LINE AS FOLLOWS
  ATTRIBUTES OF MESSAGE ENTITY AT END OF FINAL.ASSIGNMENT.NOTICE ARE:
  LIST ATTRIBUTES OF MESSAGE
  SKIP 1 OUTPUT LINE
ALWAYS
''
'RETIRN'
RETURN
END ''OF FINAL.ASSIGNMENT.NOTICE
''
''   THIS ROUTINE PRESENTLY PERFORMS FUNCTIONS TO COLLECT DATA ON HOW
''     MANY CIRCUITS FAILED TO BE ESTABLISHED.  THIS ROUTINE MAY LATER
''     BE MODIFIED TO RESCHEDULE FAILED CALLS.
''
ROUTINE FOR CKT.IS.NOT.ESTAB GIVEN NO.MSG
LET MESSAGE = NO.MSG
LET TYPE(MESSAGE) = PARTIAL.BREAKDOWN
''
LET CKT.FAILED = CKT.FAILED + 1
LET UP.ROUTE = UP.ROUTE - 1
LET DOWN.ROUTE = DOWN.ROUTE + 1
''
IF PRNT LE 4
  PRINT 2 LINES WITH CKT.NR(MESSAGE) AND TIME.V AS FOLLOWS
CIRCUIT NR. ***** CANNOT BE ESTABLISHED AND IS BEING BROKEN DOWN AT
  TIME.V = ****.****** SECONDS.
  SKIP 1 OUTPUT LINES
ALWAYS
''
RETURN
END ''OF CKT.IS.NOT.ESTAB
''
''   THIS EVENT BREAKS DOWN A ONCE ESTABLISHED CIRCUIT IN THE "UPSTREAM"
''     DIRECTION, I.E. FROM THE ORIGINATOR NODE TO THE DESTINATION NODE.
''     THIS EVENT ACTUALLY REMOVES SLOT ASSIGNMENTS FROM THE NODAL SLOT
''     ASSIGNMENT TABLES TO MAKE AVAILABLE NODE ASSETS AND CHANNEL CAPA-
''     CITY FOR USE IN THE ESTABLISHMENT OF OTHER CIRCUITS.  THIS EVENT
''     HAS TWO MAJOR SUB-SECTIONS, HOWEVER ONLY ONE OF THESE SECTIONS IS
''     EXECUTED DEPENDING ON THE VALUE OF THE "DIRECTION" ATTRIBUTE IN
''     THE "MESSAGE" ENTITY.  IF DIRECTION(MESSAGE) =
''       -2 ==>  START BREAKING DOWN AN ESTABLISHED CIRCUIT FROM THE
''                 ORIGINATOR NODE TO THE DESTINATION NODE.
''       -1 ==>  CONTINUE BREAKING DOWN A ONCE ESTABLISHED CIRCUIT FROM
''                 AN INTERMEDIATE NODE TO THE DESTINATION NODE.
''        0 ==>  RARE UPSTREAM BREAK DOWN OF A RECEIVE SLOT ASSIGNMENT
''                 LEFT HANGING AT THE CALLED.NODE WHEN A RESPONSE.REQ.-
''                 FOR.SVC FAILED.
''
EVENT UPSTREAM.BREAK.DOWN GIVEN U.B.C.MSG
```

Dro

```
LET MESSAGE = U.B.D.MSG
DEFINE INCREMENT AS A REAL VARIABLE
''
IF PRNT LE 1
  PRINT 2 LINES WITH TIME.V AS FOLLOWS
EVENT UPSTREAM.BREAK.DOWN INVOKED AT TIME.V = ****.******
-------------------------------------------------------------------------
  SKIP 1 OUTPUT LINE
  PRINT 1 LINE AS FOLLOWS
  ATTRIBUTES OF THE MESSAGE ENTITY AT START OF UPSTREAM.BREAK.DOWN ARE:
  LIST ATTRIBUTES OF MESSAGE
  SKIP 1 OUTPUT LINE
ALWAYS
''
IF TYPE(MESSAGE) EQ 2
  LET TYPE(MESSAGE) = PARTIAL.BREAKDOWN
ALWAYS
''
''
LET CURRENT.SLOT = SLOT.ARRIVAL(MESSAGE)
LET SPEC.PRINT.FLAG = 0
''
IF DIRECTION(MESSAGE) EQ -1
  GO TO CONT.BREAK.DOWN
ALWAYS
''
IF DIRECTION(MESSAGE) EQ 0
  GO TO RARE.UPSTREAM.BREAKDOWN
ALWAYS
''
IF PRNT LE 4 AND DIRECTION(MESSAGE) EQ -2 AND TYPE(MESSAGE) EQ
  FULL.BREAKDOWN
  PRINT 5 LINES WITH CKT.NR(MESSAGE), ORIGINATOR(MESSAGE), DESTINATION
    (MESSAGE), TIME.V, START.TIME(MESSAGE), ORIGINATOR(MESSAGE) AND
    DESTINATION(MESSAGE) AS FOLLOWS
CIRCUIT NR. ***** FROM NODE ** TO NODE ** WAS ONCE ESTABLISHED AND IS
  BEGINNING TO BE DISESTABLISHED AT THIS TIME, TIME.V = ****.******
  AFTER ACTIVELY CARRYING VOICE TRAFFIC FOR A TOTAL CALL DURATION OF
  ****.****** SECONDS.  BREAK DOWN IS BEING DONE FROM THE ORIGINATOR
  (NODE **) TO THE DESTINATION NODE (NODE **) IN THE UPSTREAM DIRECTION.
  SKIP 1 OUTPUT LINE
ALWAYS
''
IF PRNT LE 3 AND DIRECTION(MESSAGE) EQ -2 AND TYPE(MESSAGE) EQ
  REMOVE.LOOP
  PRINT 5 LINES WITH CKT.NR(MESSAGE), TIME.V, ORIGINATOR(MESSAGE) AND
    DESTINATION(MESSAGE) AS FOLLOWS
CIRCUIT NR. ***** HAS EXPERIENCED SOME AMOUNT OF BACKTRACKING OR LOOPING
  BACK ACROSS ITSELF.  THE LOOP IS BEING REMOVED AT THIS TIME, TIME.V =
  ****.****** AS WE START SENDING A SPECIAL "LOOP.BD.MSG" IN THE UP-
  STREAM DIRECTION FROM THE NODE THAT DISCOVERED THE LOOP (NODE **) TO
  THE LAST NODE IN THE LOOP (NODE **).
  SKIP 1 OUTPUT LINE
ALWAYS
''
LET FM.NODE(MESSAGE) = ORIGINATOR(MESSAGE)
LET START.TIME(MESSAGE) = TIME.V
IF TYPE(MESSAGE) NE REMOVE.LOOP
  LET ACTIVE = ACTIVE - 1
  LET DOWN.ROUTE = DOWN.ROUTE + 1
ALWAYS
LET DIRECTION(MESSAGE) = -1
FOR I = 1 TO SLOTS, DO
  IF USE(FM.NODE(MESSAGE),I,1) EQ CKT.NR(MESSAGE) AND
    USE(FM.NODE(MESSAGE),I,5) EQ 0
      LET SLOT2.XMIT = I
      LET TO.NODE(MESSAGE) = USE(FM.NODE(MESSAGE),I,3)
      LET M = USE(FM.NODE(MESSAGE),I,2)
      LET RECSLOT(MESSAGE) = M
      LET USE(FM.NODE(MESSAGE),M,4) = USE(FM.NODE(MESSAGE),M,4) - 1
      LET USE(FM.NODE(MESSAGE),I,1) = 0
      LET USE(FM.NODE(MESSAGE),I,2) = 0
```

```
      LET USE(FM.NODE(MESSAGE),I,3) = 0
      LET USE(FM.NODE(MESSAGE),I,5) = 0
      LET USE(FM.NODE(MESSAGE),I,6) = 0
      GO TO COMPUTE.DELAY
   ALWAYS
LOOP
PRINT 1 LINE WITH CKT.NR(MESSAGE) AS FOLLOWS
ERROR IN UPSTREAM.BREAK.DOWN FOR INITIAL BREAK DOWN OF CIRCUIT NR. *****
SKIP 1 OUTPUT LINE
GO TO REETURN
''
''    DIRECTLY ABOVE WE FOUND AND SET THE TRANSMIT AND RECEIVE SLOTS AT
''       THE ORIGINATOR NODE EQUAL TO ZERO.  DIRECTLY BELOW WE CONTINUE
''       BREAKING DOWN THE CIRCUIT ALONG THE UPSTREAM PATH.  WE FIRST
''       CHECK TO SEE IF WE ARE AT THE DESTINATION NODE, IF SO, WE NEED
''       ONLY DELETE THE TRANSMIT AND RECEIVE SLOT ASSIGNMENTS FOR THIS
''       CIRCUIT AND THEN COLLECT STATISTICS.
''
'CONT.BREAK.DOWN'
LET SLOT1.REC = SLOT.ARRIVAL(MESSAGE)
LET SLOT1.XMIT = RECSLOT(MESSAGE)
LET USE(TO.NODE(MESSAGE),SLOT1.XMIT,1) = 0
LET USE(TO.NODE(MESSAGE),SLOT1.XMIT,2) = 0
LET USE(TO.NODE(MESSAGE),SLOT1.XMIT,3) = 0
LET USE(TO.NODE(MESSAGE),SLOT1.XMIT,5) = 0
LET USE(TO.NODE(MESSAGE),SLOT1.XMIT,6) = 0
LET USE(TO.NODE(MESSAGE),SLOT1.REC,4) = USE(TO.NODE(MESSAGE),
   SLOT1.REC,4) - 1
LET CHANGE.FLAG = 1
''
''    WE HAVE NOW ERASED THE DOWN-SIDE RECEIVE AND TRANSMIT SLOT ASSIGN-
''       MENTS.
''
''    CHECK TO SEE IF WE ARE AT THE DESTINATION NODE.  IF WE ARE, THEN WE
''       HAVE ELIMINATED THE DOWN-SIDE ASSIGNMENTS AND CAN NOW COLLECT
''       STATISTICS.  OTHERWISE, CONTINUE BY BREAKING DOWN THE UP-SIDE
''       SLOT ASSIGNMENTS.
''
IF TO.NODE(MESSAGE) EQ DESTINATION(MESSAGE) AND TYPE(MESSAGE) NE
   REMOVE.LOOP
   LET START.TIME(MESSAGE) = TIME.V - START.TIME(MESSAGE)
   PERFORM COLLECT.STATS.AT.END.OF.BREAK.DOWN GIVEN MESSAGE
   GO TO REETURN
ALWAYS
IF TO.NODE(MESSAGE) EQ DESTINATION(MESSAGE) AND TYPE(MESSAGE) EQ
   REMOVE.LOOP
   LET ACT.LOOP.REMOVE = ACT.LOOP.REMOVE - 1
   DESTROY THE MESSAGE CALLED U.B.D.MSG
   GO TO REETURN
ALWAYS
LET FM.NODE(MESSAGE) = TO.NODE(MESSAGE)
FOR I = 1 TO SLOTS, DO
   IF USE(FM.NODE(MESSAGE),I,1) EQ CKT.NR(MESSAGE)
      LET SLOT2.XMIT = I
      LET TO.NODE(MESSAGE) = USE(FM.NODE(MESSAGE),I,3)
      LET M = USE(FM.NODE(MESSAGE),I,2)
      LET RECSLOT(MESSAGE) = M
      LET USE(FM.NODE(MESSAGE),M,4) = USE(FM.NODE(MESSAGE),M,4) - 1
      LET USE(FM.NODE(MESSAGE),I,1) = 0
      LET USE(FM.NODE(MESSAGE),I,2) = 0
      LET USE(FM.NODE(MESSAGE),I,3) = 0
      LET USE(FM.NODE(MESSAGE),I,5) = 0
      LET USE(FM.NODE(MESSAGE),I,6) = 0
      LET CHANGE.FLAG = 1
      GO TO COMPUTE.DELAY
   ALWAYS
LOOP
PRINT 1 LINE WITH CKT.NR(MESSAGE) AS FOLLOWS
ERROR IN UPSTREAM.BREAK.DOWN, CONTINUED BREAK DOWN OF CIRCUIT NR. *****
SKIP 1 OUTPUT LINE
GO TO REETURN
''
```

```
''   WE SHALL USE THE FORMERLY ASSIGNED TRANSMIT SLOT TO CARRY THE BREAK
''      DOWN MESSAGE TO THE NEXT NODE UPSTREAM ON THE WAY TO THE DESTINA-
''      TION NODE.  NOW CALCULATE WHEN THE BREAK DOWN MESSAGE WILL ARRIVE
''      AT THE NEXT NODE.
''
'COMPUTE.DELAY'
IF SLOT2.XMIT GT (CURRENT.SLOT + 1)
   LET DELAY = SLOT2.XMIT - CURRENT.SLOT
   GO TO SKEDULE
ALWAYS
IF SLOT2.XMIT EQ (CURRENT.SLOT + 1)
   LET DELAY = SLOTS
   GO TO SKEDULE
ALWAYS
IF SLOT2.XMIT LT (CURRENT.SLOT + 1)
   LET DELAY = (SLOT2.XMIT + SLOTS - CURRENT.SLOT)
   GO TO SKEDULE
ALWAYS
PRINT 1 LINE WITH CKT.NR(MESSAGE) AS FOLLOWS
ERROR IN UPSTREAM.BREAK.DOWN, DELAY CALCULATION FOR CIRCUIT NR. = *****
SKIP 1 OUTPUT LINE
GO TO REETURN
''
'SKEDULE'
LET SLOT.ARRIVAL(MESSAGE) = SLOT2.XMIT
LET INCREMENT = REAL.F(DELAY) * SLOT.DURATION
SCHEDULE AN UPSTREAM.BREAK.DOWN GIVEN MESSAGE IN INCREMENT UNITS
GO TO REETURN
''
'RARE.UPSTREAM.BREAKDOWN'
LET USE(TO.NODE(MESSAGE),SLOT.ASSIGN(MESSAGE),4) = USE(TO.NODE(MESSAGE),
   SLOT.ASSIGN(MESSAGE),4) - 1
LET CHANGE.FLAG = 1
IF RECSLOT(MESSAGE) EQ SLOTS + 1
   LET START.TIME(MESSAGE) = TIME.V - START.TIME(MESSAGE)
   PERFORM COLLECT.STATS.AT.END.OF.BREAK.DOWN GIVEN MESSAGE
ALWAYS
IF RECSLOT(MESSAGE) LE SLOTS
   DESTROY THE MESSAGE CALLED U.B.D.MSG
ALWAYS
LET SPEC.PRINT.FLAG = 1
GO TO REETURN
''
'REETURN'
IF PRNT LE 1 AND SPEC.PRINT.FLAG EQ 0
   PRINT 1 LINE AS FOLLOWS
   ATTRIBUTES OF THE MESSAGE ENTITY AT END OF UPSTREAM.BREAK.DOWN ARE:
   LIST ATTRIBUTES OF MESSAGE
   SKIP 1 OUTPUT LINE
ALWAYS
''
RETURN
END ''OF UPSTREAM.BREAK.DOWN
''
''   THIS EVENT BREAKS DOWN SOME FULLY ESTABLISHED AND ALL PARTIALLY ES-
''      TABLISHED CIRCUITS.  BREAK DOWN IS PERFORMED IN THE "DOWNSTREAM"
''      DIRECTION, I.E. FROM THE DESTINATION OR FURTHEST NODE REACHED
''      BACK TO THE ORIGINATOR NODE.  THIS EVENT HAS SEVERAL SUB-SECTIONS
''      AND EACH TIME IT IS EXECUTED ONLY ONE OF THE MAJOR SECTIONS IS
''      EXECUTED ACCORDING TO THE VALUE OF THE "DESTINATION" ATTRIBUTE OF
''      THE "MESSAGE" ENTITY.  IF DIRECTION(MESSAGE) =
''         +1 ==>  START BREAKING DOWN AN ESTABLISHED CIRCUIT FROM THE
''                    DESTINATION NODE TO THE ORIGINATOR NODE.
''         +2 ==>  CONTINUE BREAKING DOWN A ONCE ESTABLISHED OR PARTIALLY
''                    ESTABLISHED CIRCUIT FROM AN INTERMEDIATE NODE TO THE
''                    ORIGINATOR NODE.
''         +3 ==>  START BREAKING DOWN A PARTIALLY ESTABLISHED CIRCUIT
''                    FROM THE FURTHEST NODE REACHED TO THE ORIGINATOR
''                    NODE.  CALLED BY INITIAL.REC.FOR.SVC.
''         +4 ==>  START BREAKING DOWN A PARTIALLY ESTABLISHED CIRCUIT
''                    FROM THE FURTHEST NODE REACHED TO THE ORIGINATOR
''                    NODE.  CALLED BY RESPONSE.REC.FOR.SVC.
```

```
''        +5 ==>  SPECIAL CASE BREAK DOWN OF A PARTIALLY ESTABLISHED CIR-
''                    CUIT.  CALLED BY FINAL.ASSIGNMENT.NOTICE.
''
EVENT DOWNSTREAM.BREAK.DOWN GIVEN D.B.D.MSG
LET MESSAGE = D.B.D.MSG
DEFINE INCREMENT AS A REAL VARIABLE
''
IF PRNT LE 1
  PRINT 2 LINES WITH TIME.V AS FOLLOWS
EVENT DOWNSTREAM.BREAK.DOWN INVOKED AT TIME.V = ****.******
-------------------------------------------------------------
  SKIP 1 OUTPUT LINE
  PRINT 1 LINE AS FOLLOWS
  ATTRIBUTES OF THE MESSAGE ENTITY AT START OF DOWNSTREAM.BREAK.DOWN ARE
  LIST ATTRIBUTES OF MESSAGE
  SKIP 1 OUTPUT LINE
ALWAYS
''
IF TYPE(MESSAGE) EQ 2
  LET TYPE(MESSAGE) = PARTIAL.BREAKDOWN
ALWAYS
''
LET CURRENT.SLOT = SLOT.ARRIVAL(MESSAGE)
LET OPT.PRINT.FLAG = 0
''
IF DIRECTION(MESSAGE) EQ 1
  GO TO FIRST.LABEL
ALWAYS
IF DIRECTION(MESSAGE) EQ 2
  GO TO SECOND.LABEL
ALWAYS
IF DIRECTION(MESSAGE) EQ 3
  GO TO THIRD.LABEL
ALWAYS
IF DIRECTION(MESSAGE) EQ 4
  GO TO FOURTH.LABEL
ALWAYS
IF DIRECTION(MESSAGE) EQ 5
  GO TO FIFTH.LABEL
ALWAYS
''
''    ACTIONS PERFORMED UNDER THE FIRST LABEL SIMULATE THE START OF DIS-
''       ESTABLISHMENT OF A CIRCUIT THAT WAS ONCE ESTABLISHED AND ACTIVE.
''
'FIRST.LABEL'
IF PRNT LE 4 AND DIRECTION(MESSAGE) EQ 1
  PRINT 5 LINES WITH CKT.NR(MESSAGE), ORIGINATOR(MESSAGE),
     DESTINATION(MESSAGE), TIME.V, START.TIME(MESSAGE),
     DESTINATION(MESSAGE) AND ORIGINATOR(MESSAGE) AS FOLLOWS
CIRCUIT NR. ***** FROM NODE ** TO NODE ** WAS ONCE ESTABLISHED AND IS
  BEGINNING TO BE DISESTABLISHED AT THIS TIME, TIME.V = ****.******
  AFTER ACTIVELY CARRYING VOICE TRAFFIC FOR A TOTAL CALL DURATION OF
  ****.****** SECONDS.  BREAK DOWN IS BEING DONE FROM THE DESTINATION
  (NODE **) TO THE ORIGINATOR (NODE **) IN THE DOWNSTREAM DIRECTION.
  SKIP 1 OUTPUT LINE
ALWAYS
LET FM.NODE(MESSAGE) = DESTINATION(MESSAGE)
LET START.TIME(MESSAGE) = TIME.V
LET ACTIVE = ACTIVE - 1
LET DOWN.ROUTE = DOWN.ROUTE + 1
LET DIRECTION(MESSAGE) = 2
''
'JUMP.IN'
FOR I = 1 TO SLOTS, DO
  IF USE(FM.NODE(MESSAGE),I,1) EQ CKT.NR(MESSAGE)
     LET SLOT1.XMIT = I
     LET TO.NODE(MESSAGE) = USE(FM.NODE(MESSAGE),I,3)
     LET M = USE(FM.NODE(MESSAGE),I,2)
     LET RECSLOT(MESSAGE) = M
     LET USE(FM.NODE(MESSAGE),M,4) = USE(FM.NODE(MESSAGE),M,4) - 1
     LET USE(FM.NODE(MESSAGE),I,1) = 0
     LET USE(FM.NODE(MESSAGE),I,2) = 0
```

```
      LET USE(FM.NODE(MESSAGE),I,3) = 0
      LET USE(FM.NODE(MESSAGE),I,5) = 0
      LET USE(FM.NODE(MESSAGE),I,6) = 0
      LET CHANGE.FLAG = 1
      GO TO CALCULATE.DELAY
    ALWAYS
LOOP
PRINT 1 LINE WITH CKT.NR(MESSAGE) AS FOLLOWS
ERROR IN DOWNSTREAM.BREAK.DOWN, FIRST.LABEL FOR CIRCUIT NUMBER = *****
SKIP 1 OUTPUT LINE
GO TO REEETURN
''
''    DIRECTLY ABOVE WE FOUND AND DELETED THE DOWN-SIDE TRANSMIT AND
''       RECEIVE SLOT ASSIGNMENTS AT THE DESTINATION NODE OR THE FURTHEST
''       NODE REACHED.  IN THE SECTION LABELLED "SECOND.LABEL" BELOW WE
''       CONTINUE BREAKING DOWN THE CIRCUIT ALONG THE DOWNSTREAM PATH.
''
'SECOND.LABEL'
LET SLOT2.REC = SLOT.ARRIVAL(MESSAGE)
LET SLOT2.XMIT = RECSLCT(MESSAGE)
LET USE(TO.NODE(MESSAGE),SLOT2.XMIT,1) = 0
LET USE(TO.NODE(MESSAGE),SLOT2.XMIT,2) = 0
LET USE(TO.NODE(MESSAGE),SLOT2.XMIT,3) = 0
LET USE(TO.NODE(MESSAGE),SLOT2.XMIT,5) = 0
LET USE(TO.NODE(MESSAGE),SLOT2.XMIT,6) = 0
LET USE(TO.NODE(MESSAGE),SLOT2.REC,4) = USE(TO.NODE(MESSAGE),
   SLOT2.REC,4) - 1
LET CHANGE.FLAG = 1
''
''    WE HAVE NOW ERASED THE UP-SIDE RECEIVE AND TRANSMIT SLOT ASSIGN-
''       MENTS.
''
''    CHECK TO SEE IF WE ARE AT THE ORIGINATOR NODE.  IF WE ARE, THEN WE
''       HAVE ELIMINATED THE UP-SIDE ASSIGNMENTS AND CAN NOW COLLECT
''       STATISTICS.  OTHERWISE, CONTINUE BY BREAKING DOWN THE DOWN-SIDE
''       SLOT ASSIGNMENTS.
''
IF TO.NODE(MESSAGE) EQ ORIGINATOR(MESSAGE)
   LET START.TIME(MESSAGE) = TIME.V - START.TIME(MESSAGE)
   PERFORM COLLECT.STATS.AT.END.OF.BREAK.DOWN GIVEN MESSAGE
   LET OPT.PRINT.FLAG = 1
   GO TO REEETURN
ALWAYS
LET FM.NODE(MESSAGE) = TO.NODE(MESSAGE)
FOR I = 1 TO SLOTS, DO
   IF USE(FM.NODE(MESSAGE),I,1) EQ CKT.NR(MESSAGE)
      LET SLOT1.XMIT = I
      LET TO.NODE(MESSAGE) = USE(FM.NODE(MESSAGE),I,3)
      LET M = USE(FM.NODE(MESSAGE),I,2)
      LET RECSLCT(MESSAGE) = M
      LET USE(FM.NODE(MESSAGE),M,4) = USE(FM.NODE(MESSAGE),M,4) - 1
      LET USE(FM.NODE(MESSAGE),I,1) = 0
      LET USE(FM.NODE(MESSAGE),I,2) = 0
      LET USE(FM.NODE(MESSAGE),I,3) = 0
      LET USE(FM.NODE(MESSAGE),I,5) = 0
      LET USE(FM.NODE(MESSAGE),I,6) = 0
      LET CHANGE.FLAG = 1
      GO TO CALCULATE.DELAY
    ALWAYS
LOOP
PRINT 1 LINE WITH CKT.NR(MESSAGE) AS FOLLOWS
ERROR IN DOWNSTREAM.BREAK.DOWN, SECOND.LABEL, CIRCUIT NUMBER = *****
SKIP 1 OUTPUT LINE
GO TO REEETURN
''
''    AT THE "THIRD.LABEL" WE START BREAKING DOWN A PARTIALLY ESTABLISHED
''       CIRCUIT FROM THE FURTHEST NODE REACHED.  THIS PART OF THE EVENT
''       IS EXECUTED AS A RESULT OF INITIATING A BREAK DOWN FROM THE "INI-
''       TIAL.REQ.FOR.SVC" EVENT.
''
'THIRD.LABEL'
IF PRNT LE 1 AND DIRECTION(MESSAGE) EQ 3
```

```
   PRINT 5 LINES WITH CKT.NR(MESSAGE), ORIGINATOR(MESSAGE), DESTINATION
      (MESSAGE), TIME.V, START.TIME(MESSAGE) AND HOP.COUNT(MESSAGE)
      AS FOLLOWS
CIRCUIT NR. ***** FROM NODE ** TO NODE ** CANNOT BE ESTABLISHED.  THE
   TIME NOW IS TIME.V = ****.******, AND WE BEGAN BREAKING DOWN THE CIR-
   CUIT AT TIME.V = ****.****** BY RELEASING SLOT ASSIGNMENTS ON A LINK
   BY LINK BASIS BACK TO THE ORIGINATOR NODE.  **.* HOPS WERE ESTABLISHED
   BEFORE THE CIRCUIT FAILED AND BREAK DOWN BEGAN.  IRFS3
   SKIP 1 OUTPUT LINE
ALWAYS
LET DIRECTION(MESSAGE) = 2
GO TO JUMP.IN
''
''    AT THE "FOURTH.LABEL" WE ALSO BEGIN BREAKING DOWN A PARTIALLY
''       ESTABLISHED CIRCUIT FROM THE FARTHEST NODE REACHED.  THIS PART OF
''       THE EVENT IS EXECUTED AS A RESULT OF INITIATING A BREAK DOWN FROM
''       THE "RESPONSE.REQ.FOR.SVC" EVENT.
''
'FOURTH.LABEL'
IF PRNT LE 1 AND DIRECTION(MESSAGE) EQ 4
   PRINT 5 LINES WITH CKT.NR(MESSAGE), ORIGINATOR(MESSAGE), DESTINATION
      (MESSAGE), TIME.V, START.TIME(MESSAGE) AND HOP.COUNT(MESSAGE)
      AS FOLLOWS
CIRCUIT NR. ***** FROM NODE ** TO NODE ** CANNOT BE ESTABLISHED.  THE
   TIME NOW IS TIME.V = ****.******, AND WE BEGAN BREAKING DOWN THE CIR-
   CUIT AT TIME.V = ****.****** BY RELEASING SLOT ASSIGNMENTS ON A LINK
   BY LINK BASIS BACK TO THE ORIGINATOR NODE.  **.* HOPS WERE ESTABLISHED
   BEFORE THE CIRCUIT FAILED AND BREAK DOWN BEGAN.  RRFS4
   SKIP 1 OUTPUT LINE
ALWAYS
LET DIRECTION(MESSAGE) = 2
GO TO JUMP.IN
''
'FIFTH.LABEL'
IF PRNT LE 1 AND DIRECTION(MESSAGE) EQ 5
   PRINT 5 LINES WITH CKT.NR(MESSAGE), ORIGINATOR(MESSAGE), DESTINATION
      (MESSAGE), TIME.V, START.TIME(MESSAGE) AND HOP.COUNT(MESSAGE)
      AS FOLLOWS
CIRCUIT NR. *****, FROM NODE ** TO NODE ** CANNOT BE ESTABLISHED.  THE
   TIME NOW IS TIME.V = ****.******, AND WE BEGAN BREAKING DOWN THE CIR-
   CUIT AT TIME.V = ****.****** BY RELEASING SLOT ASSIGNMENTS ON A LINK
   BY LINK BASIS BACK TO THE ORIGINATOR NODE.  **.* HOPS WERE ESTABLISHED
   BEFORE THE CIRCUIT FAILED AND BREAK DOWN BEGAN.  F.A.N CONTENTION.
   SKIP 1 OUTPUT LINE
ALWAYS
LET USE(FM.NODE(MESSAGE),SLOT.ASSIGN(MESSAGE),1) = 0
LET USE(FM.NODE(MESSAGE),SLOT.ASSIGN(MESSAGE),2) = 0
LET USE(FM.NODE(MESSAGE),SLOT.ASSIGN(MESSAGE),3) = 0
LET USE(FM.NODE(MESSAGE),SLOT.ASSIGN(MESSAGE),5) = 0
LET USE(FM.NODE(MESSAGE),SLOT.ASSIGN(MESSAGE),6) = 0
LET USE(FM.NODE(MESSAGE),RECSLOT(MESSAGE),4) = USE (FM.NODE(MESSAGE),
   RECSLOT(MESSAGE),4) - 1
LET CHANGE.FLAG = 1
IF FM.NODE(MESSAGE) EQ ORIGINATOR(MESSAGE)
   LET START.TIME(MESSAGE) = TIME.V - START.TIME(MESSAGE)
   PERFORM COLLECT.STATS.AT.END.OF.BREAK.DOWN GIVEN MESSAGE
   LET OPT.PRINT.FLAG = 1
   GO TO REEETURN
ALWAYS
LET DIRECTION(MESSAGE) = 2
GO TO JUMP.IN
''
'CALCULATE.DELAY'
IF SLOT1.XMIT GT (CURRENT.SLOT + 1)
   LET DELAY = SLOT1.XMIT - CURRENT.SLOT
   GO TO SKEDULE
ALWAYS
IF SLOT1.XMIT EQ (CURRENT.SLOT + 1)
   LET DELAY = SLOTS + 1
   GO TO SKEDULE
ALWAYS
IF SLOT1.XMIT LT (CURRENT.SLOT + 1)
```

```
   LET DELAY = (SLOT1.XMIT + SLOTS - CURRENT.SLOT)
   GO TO SKEDULE
ALWAYS
PRINT 1 LINE WITH CKT.NR(MESSAGE) AS FOLLOWS
ERROR IN DOWNSTREAM.BREAK.DOWN, DELAY CALCULATION FOR CIRCUIT NR. *****
SKIP 1 OUTPUT LINE
GO TO REEETURN
''
'SKEDULE'
LET SLOT.ARRIVAL(MESSAGE) = SLOT1.XMIT
LET INCREMENT = REAL.F(DELAY) * SLOT.DURATION
SCHEDULE A DOWNSTREAM.BREAK.DOWN GIVEN MESSAGE IN INCREMENT UNITS
''
'REEETURN'
IF PRNT LE 1 AND OPT.PRINT.FLAG = 0
   PRINT 1 LINE AS FOLLOWS
   ATTRIBUTES OF THE MESSAGE ENTITY AT END OF DOWNSTREAM.BREAK.DOWN ARE:
   LIST ATTRIBUTES OF MESSAGE
   SKIP 1 OUTPUT LINE
ALWAYS
''
RETURN
END ''OF DOWNSTREAM.BREAK.DOWN
''
''    THIS ROUTINE COLLECTS DATA ON CIRCUITS THAT ARE ESTABLISHED AND
''       SCHEDULES THEIR EVENTUAL DISESTABLISHMENT ACCORDING TO AN EXPO-
''       NENTIAL DISTRIBUTION FUNCTION WITH A "MEAN.DURATION.OF.CKT" GIVEN
''       AS AN INPUT PARAMETER IN THE ROUTINE FOR INITIALIZATION.
''
ROUTINE FOR COMPLETED.CKT GIVEN ARRIVAL.MSG
LET MESSAGE = ARRIVAL.MSG
''
IF PRNT LE 1
   PRINT 2 LINES WITH TIME.V AS FOLLOWS
ROUTINE COMPLETED.CKT CALLED AT TIME.V = ****.******
------------------------------------------------------
   SKIP 1 OUTPUT LINE
ALWAYS
IF PRNT LE 3
   PRINT 1 LINE AS FOLLOWS
   ATTRIBUTES OF THE MESSAGE ENTITY WHEN COMPLETED.CKT WAS CALLED ARE:
   LIST ATTRIBUTES OF MESSAGE
   SKIP 1 OUTPUT LINE
ALWAYS
''
LET CKT.ESTAB = CKT.ESTAB + 1
LET UP.ROUTE = UP.ROUTE - 1
LET ACTIVE = ACTIVE + 1
LET CHANGE.FLAG = 1
IF HOP.COUNT(MESSAGE) EQ HOP.GREATEST
   LET TOT.HOP.GREATEST = TOT.HOP.GREATEST + 1
   LET CKT.GREATEST = CKT.NR(MESSAGE)
ALWAYS
IF HOP.COUNT(MESSAGE) GT HOP.GREATEST
   LET HOP.GREATEST = HOP.COUNT(MESSAGE)
   LET TOT.HOP.GREATEST = 1
   LET CKT.GREATEST = CKT.NR(MESSAGE)
ALWAYS
LET HOP.SUM = HOP.SUM + HOP.COUNT(MESSAGE)
LET HOP.AVG = HOP.SUM / REAL.F(CKT.ESTAB)
LET DELAY.SUM = DELAY.SUM + START.TIME(MESSAGE)
LET E.SUM = E.SUM + CUM.ENERGY(MESSAGE)
LET E.SUB.K.BAR = E.SUM / REAL.F(CKT.ESTAB)
LET AVG.TIME.EST = DELAY.SUM / REAL.F(CKT.ESTAB)
''
''    DETERMINE IF THIS CIRCUIT TOOK THE MOST TIME OF ANY CIRCUIT TO
''       ESTABLISH.
''
IF START.TIME(MESSAGE) GT LONG.TIME.EST
   LET LONG.TIME.EST = START.TIME(MESSAGE)
   LET CKT.LONG.TIME.EST = CKT.NR(MESSAGE)
ALWAYS
```

```
''
''   COLLECT LINK USAGE STATISTICS FOR THIS CIRCUIT.
''
LET LIN.K.USED(INFO1(MESSAGE)) = LIN.K.USED(INFO1(MESSAGE)) + 1
IF INFO2(MESSAGE) EQ 0
  GO TO GET.DURATION
ALWAYS
LET LIN.K.USED(INFO2(MESSAGE)) = LIN.K.USED(INFO2(MESSAGE)) + 1
IF INFO3(MESSAGE) EQ 0
  GO TO GET.DURATION
ALWAYS
LET LIN.K.USED(INFO3(MESSAGE)) = LIN.K.USED(INFO3(MESSAGE)) + 1
IF INFO4(MESSAGE) EQ 0
  GO TO GET.DURATION
ALWAYS
LET LIN.K.USED(INFO4(MESSAGE)) = LIN.K.USED(INFO4(MESSAGE)) + 1
IF INFO5(MESSAGE) EQ 0
  GO TO GET.DURATION
ALWAYS
LET LIN.K.USED(INFO5(MESSAGE)) = LIN.K.USED(INFO5(MESSAGE)) + 1
IF INFO6 EQ 0
  GO TO GET.DURATION
ALWAYS
LET LIN.K.USED(INFO6(MESSAGE)) = LIN.K.USED(INFO6(MESSAGE)) + 1
IF INFO7 EQ 0
  GO TO GET.DURATION
ALWAYS
LET LIN.K.USED(INFO7(MESSAGE)) = LIN.K.USED(INFO7(MESSAGE)) + 1
IF INFO8 EQ 0
  GO TO GET.DURATION
ALWAYS
LET LIN.K.USED(INFO8(MESSAGE)) = LIN.K.USED(INFC8(MESSAGE)) + 1
IF INFO9 EQ 0
  GO TO GET.DURATION
ALWAYS
LET LIN.K.USED(INFO9(MESSAGE)) = LIN.K.USED(INFC9(MESSAGR)) + 1
''
'GET.DURATION'
''
''   DETERMINE HOW LONG THIS CIRCUIT WHICH WAS JUST ESTABLISHED WILL BE
''      "ACTIVE" THEN SELECT FROM WHICH NODE (ORIGINATOR OR DESTINATION)
''      THE CIRCUIT WILL BE DISESTABLISHED AND SCHEDULE THE EVENT TO
''      BREAK DOWN THIS CIRCUIT.
''
LET DURATION = EXPONENTIAL.F(MEAN.DURATION.OF.CKT,3)
LET SUM.DURATION = SUM.DURATION + DURATION
LET AVG.DURATION = SUM.DURATION / REAL.F(CKT.ESTAB)
LET START.TIME(MESSAGE) = DURATION
LET TYPE(MESSAGE) = FULL.BREAKDOWN
''
''   RANDOMLY SELECT AND STORE A "CURRENT.SLOT" THAT WE WILL ASSUME TO
''      BE IN WHEN THIS CIRCUIT IS EVENTUALLY BROKEN DOWN.
''
LET SLOT.ARRIVAL(MESSAGE) = RANDI.F(1,SLOTS,5)
IF PRNT LE 1
  PRINT 2 LINES WITH SLOT.ARRIVAL(MESSAGE) AS FOLLOWS
  SLOT ** WAS RANDOMLY SELECTED AS THE "CURRENT.SLOT" WHICH WILL BE THE
     SLOT WE ARE IN WHEN WE EVENTUALLY BEGIN BREAKING DOWN THIS CIRCUIT.
  SKIP 1 OUTPUT LINE
ALWAYS
''
''   THE VALUE OF STARTER IS SET IN THE INITIALIZATION ROUTINE.
''
IF STARTER EQ 1
  LET STARTER = 0
  LET FM.NODE(MESSAGE) = ORIGINATOR(MESSAGE)
  LET DIRECTION(MESSAGE) = -2
  SCHEDULE AN UPSTREAM.BREAK.DOWN GIVEN MESSAGE IN DURATION UNITS
  GO TO FINIS
ALWAYS
IF STARTER EQ 0
  LET STARTER = 1
```

```
   LET DIRECTION(MESSAGE) = 1
   SCHEDULE A DOWNSTREAM.BREAK.DOWN GIVEN MESSAGE IN DURATION UNITS
   GO TO FINIS
ALWAYS
''
'FINIS'
IF PRNT LE 2 AND STARTER EQ 0
   PRINT 4 LINES WITH CKT.NR(MESSAGE), ORIGINATOR(MESSAGE),
     DESTINATION(MESSAGE), TIME.V, DURATION AND (TIME.V + DURATION)
     AS FOLLOWS
CIRCUIT NR. *****, FROM NODE ** TO NODE ** WAS ESTABLISHED AT TIME.V =
   ****.****** AND IS SCHEDULED TO LAST FOR A TOTAL CALL DURATION OF
   ****.****** SECONDS, SO BREAKDOWN WILL COMMENCE IN THE UPSTREAM DIREC-
   TION AT TIME.V = ****.******.
   SKIP 1 OUTPUT LINE
ALWAYS
IF PRNT LE 2 AND STARTER EQ 1
   PRINT 4 LINES WITH CKT.NR(MESSAGE), ORIGINATOR(MESSAGE),
   DESTINATION(MESSAGE), TIME.V, DURATION AND (TIME.V + DURATION)
   AS FOLLOWS
CIRCUIT NR. *****, FROM NODE ** TO NODE ** WAS ESTABLISHED AT TIME.V =
   ****.****** AND IS SCHEDULED TO LAST FOR A TOTAL CALL DURATION OF
   ****.****** SECONDS, SO BREAKDOWN WILL COMMENCE IN THE DOWNSTREAM
   DIRECTION AT TIME.V = ****.******.
   SKIP 1 OUTPUT LINE
ALWAYS
IF PRNT LE 1
   PRINT 1 LINE AS FOLLOWS
   ATTRIBUTES OF THE MESSAGE ENTITY AT THE END OF COMPLETED.CKT ARE:
   LIST ATTRIBUTES OF MESSAGE
   SKIP 1 OUTPUT LINE
ALWAYS
''
IF PRNT EQ 4 AND PRT LE 1 AND SPECIFY.OUTPUT EQ 0
   PRINT 1 LINE WITH CKT.NR(MESSAGE), HOP.COUNT(MESSAGE) AND TIME.V
     AS FOLLOWS
CIRCUIT NR. ***** ESTABLISHED IN **.* HOPS AT TIME.V = ****.******
   SKIP 1 OUTPUT LINE
ALWAYS
RETURN
END ''OF COMPLETED.CKT
''
''   THIS ROUTINE INCREMENTS COUNTERS AND COLLECTS STATISTICS ON THE
''     CIRCUITS THAT ARE BROKEN DOWN.  THIS ROUTINE IS ONLY CALLED BY
''     THE "UPSTREAM.BREAK.DOWN" AND "DOWNSTREAM.BREAK.DOWN" EVENTS.
''
ROUTINE ~~TO~~ FOR COLLECT.STATS.AT.END.~~OF~~.BREAK.DOWN GIVEN ~~BRK.DN.NOTICE~~ B.D.Message
LET MESSAGE = BRK.DN.NOTICE
DEFINE TIME.BD.THIS.CKT AS A REAL VARIABLE
''
IF TYPE(MESSAGE) EQ FULL.BREAKDOWN
  LET CKT.DISESTAB = CKT.DISESTAB + 1
ALWAYS
''
LET CHANGE.FLAG = 1
LET CKTS.BD = CKT.DISESTAB + CKT.FAILED
LET DOWN.ROUTE = DOWN.ROUTE - 1
LET TIME.BD.THIS.CKT = START.TIME(MESSAGE)
LET SUM.BD.TIME.ALL.CKT = SUM.BD.TIME.ALL.CKT + TIME.BD.THIS.CKT
LET AVG.BD.TIME = SUM.BD.TIME.ALL.CKT / REAL.F(CKTS.BD)
''
''   COLLECT STATS ON THE BREAK DOWN OF PARTIALLY ESTABLISHED CIRCUITS.
''
IF TYPE(MESSAGE) EQ 3
   IF START.TIME(MESSAGE) GT LONG.P.BD
     LET LONG.P.BD = START.TIME(MESSAGE)
   ALWAYS
   LET TOT.P.BD = TOT.P.BD + START.TIME(MESSAGE)
   LET P.BD.COUNTER = P.BD.COUNTER + 1
   LET AVG.P.BD = TOT.P.BD / REAL.F(P.BD.COUNTER)
ALWAYS
''
```

```
''   COLLECT STATS ON THE BREAK DOWN OF ONCE ESTABLISHED CIRCUITS.
''
IF TYPE(MESSAGE) EQ 4
   IF START.TIME(MESSAGE) GT LONG.C.BD
      LET LONG.C.BD = START.TIME(MESSAGE)
   ALWAYS
   LET TOT.C.BD = TOT.C.BD + START.TIME(MESSAGE)
   LET C.BD.COUNTER = C.BD.COUNTER + 1
   LET AVG.C.BD = TOT.C.BD / REAL.F(C.BD.COUNTER)
ALWAYS
''
DESTROY THE MESSAGE CALLED BRK.DN.NOTICE
''
RETURN
END ''OF COLLECT.STATS.AT.END.OF.BREAK.DOWN
''
''    THIS ROUTINE CONTAINS THE SPECIAL OUTPUT INFORMATION SPECIFIED BY
''       THE PROGRAMMER.  THIS ROUTINE IS ONLY EXECUTED AFTER THE SIMULA-
''       TION HAS COMPLETED ALL FOUR QUARTERS OF ONE SIMULATION RUN AND
''       THE "SPECIFY.OUTPUT" VARIABLE IS GREATER THAN OR EQUAL TO 1.
''
ROUTINE FOR SPECIAL.OUTPUT
''
PRINT 1 LINE AS FOLLOWS
THE ROUTINE FOR "SPECIAL.OUTPUT" HAS BEEN INVOKED.
SKIP 1 OUTPUT LINE
''
RETURN
END ''OF SPECIAL.OUTPUT
''
''    THE FOLLOWING ROUTINE CANCELS AND/OR DESTROYS ALL ENTITIES AND
''       EVENTS WHICH ARE CONTAINED IN THE TIMING ROUTINE AFTER TIME.V
''       EQUALS THE TEST DURATION TIME LIMIT OR AFTER THE TOTAL NUMBER OF
''       CIRCUITS ATTEMPTED EXCEEDS THE PERMITTED MAXIMUM NUMBER OF CIR-
''       CUITS IN EACH ITERATION OF THE SIMULATION.
''
ROUTINE FOR DESTRUCTION
''
FOR EACH NEW.CKT.REQMT IN EV.S(I.NEW.CKT.REQMT), DO
   CANCEL THE NEW.CKT.REQMT
   DESTROY THE NEW.CKT.REQMT
LOOP
''
FOR EACH INITIAL.REQ.FOR.SVC IN EV.S(I.INITIAL.REQ.FOR.SVC), DO
   CANCEL THE INITIAL.REQ.FOR.SVC
   DESTROY THE INITIAL.REQ.FOR.SVC
LOOP
''
FOR EACH RESPONSE.REQ.FOR.SVC IN EV.S(I.RESPONSE.REQ.FOR.SVC), DO
   CANCEL THE RESPONSE.REQ.FOR.SVC
   DESTROY THE RESPONSE.REQ.FOR.SVC
LOOP
''
FOR EACH FINAL.ASSIGNMENT.NOTICE IN EV.S(I.FINAL.ASSIGNMENT.NOTICE), DO
   CANCEL THE FINAL.ASSIGNMENT.NOTICE
   DESTROY THE FINAL.ASSIGNMENT.NOTICE
LOOP
''
FOR EACH UPSTREAM.BREAK.DOWN IN EV.S(I.UPSTREAM.BREAK.DOWN), DO
   CANCEL THE UPSTREAM.BREAK.DOWN
   DESTROY THE UPSTREAM.BREAK.DOWN
LOOP
''
FOR EACH DOWNSTREAM.BREAK.DOWN IN EV.S(I.DOWNSTREAM.BREAK.DOWN), DO
   CANCEL THE DOWNSTREAM.BREAK.DOWN
   DESTROY THE DOWNSTREAM.BREAK.DOWN
LOOP
''
FOR EACH STOP.SIMULATION IN EV.S(I.STOP.SIMULATION), DO
   CANCEL THE STOP.SIMULATION
   DESTROY THE STOP.SIMULATION
LOOP
```

```
''
FOR EACH DIJK.MANIPULATION IN EV.S(I.CIJK.MANIPULATION), DO
   CANCEL THE DIJK.MANIPULATION
   DESTROY THE DIJK.MANIPULATION
LOOP
''
FOR EACH RE.MOVE.TRANSIENT.EFFECT IN EV.S(I.RE.MOVE.TRANSIENT.EFFECT),DO
   CANCEL THE RE.MOVE.TRANSIENT.EFFECT
   DESTROY THE RE.MOVE.TRANSIENT.EFFECT
LOOP
''
RETURN
END ''OF DESTRUCTION
/*
//
```

SAMPLE INPUT DATA

```
FILE: THESIS   DATA     A1  NAVAL POSTGRADUATE SCHOOL

//T10STAT JOB (1966,0132),'TRITCHLER 1642',CLASS=C
//*MAIN ORG=NPGVM1.1966P,LINES=(5)
//*FORMAT PR,DDNAME=,DEST=LOCAL
//GO      EXEC  PGM=LOADER,PARM='MAP,SIZE=560K',REGION=1024K
//SYSLIB    DD  DSN=SYS3.SIMLIB8H,UNIT=3350,VOL=SER=MVS003,DISP=SHR
//SYSLOUT   DD  SYSOUT=*,DCB=(RECFM=FBA,LRECL=121,BLKSIZE=1210),
//              SPACE=(TRK,(1,1))
//SYSLIN    DD  DISP=SHR,DSN=MSS.S1966.THESIX.LOADLIB
//SIMU05    DD  DDNAME=SYSIN
//SIMU06    DD  SYSOUT=*.DCB=(RECFM=FBA,LRECL=133,BLKSIZE=3325)
//SIMU17    DD  DSN=SYS3.SIMERR8H,UNIT=335C,VOL=SER=MVS003,DISP=SHR
//SYSIN     DD  *
0                       <- SPECIFY.OUTPUT PRINTING VARIABLE
5                       <- PRNT DIAGNOSTIC PRINTING VARIABLE
3                       <- PRT DIAGNOSTIC PRINTING VARIABLE
1                       <- LTD.PRINT PRINTING VARIABLE
2                       <- ROUTING.ALGORITHM.SELECTOR
13                      <- N.NODE = NUMBER OF NODES IN THE NETWORK
1.0    1.0    1    1    <- TRANSMIT.PERCENT,
1.0    1.0    1    1          RECEIVE.PERCENT,
1.0    1.0    1    1             GROUP, AND FAMILY
1.0    1.0    1    1
1.0    1.0    1    1
1.0    1.0    1    1
1.0    1.0    1    1
1.0    1.0    1    1
1.0    1.0    1    1
1.0    1.0    1    1
1.0    1.0    1    1
1.0    1.0    1    1
1.0    1.0    1    1
0 1 1 1 1 0 0 0 0 0 0 0 0  <- THIS 13 BY 13 BLOCK OF 1'S AND 0'S
1 0 1 0 0 1 0 0 1 0 0 0 0        IS THE LINKABLE ARRAY USED TO
1 1 0 1 0 1 1 0 0 0 0 0 0           IDENTIFY DIRECTLY CONNECTED NODES.
1 0 1 0 1 0 1 1 0 0 0 0 0
1 0 0 1 0 0 0 1 0 0 0 1 0
0 1 1 0 0 0 1 0 1 1 0 0 0
0 0 1 1 0 1 0 1 0 1 1 0 0
0 0 0 1 1 0 1 0 0 0 1 1 0
0 1 0 0 0 1 0 0 0 1 0 0 1
0 0 0 0 0 1 1 0 1 0 1 0 1
0 0 0 0 0 0 1 1 0 1 0 1 1
0 0 0 0 1 0 0 1 0 0 1 0 1
0 0 0 0 0 0 0 0 1 1 1 1 0
 7  11      2   6      1   3      3   4      3   7  <- 30 PAIRS OF DIRECTLY
 6   9      4   8      7  10     11  12      3   6       CONNECTED NODES
 6   7      7   8      2   3      8  11      5   8
 6  10      9  13     10  13      1   2      4   5
 2   9      4   7     11  13      9  10      1   5
 8  12     10  11      1   4      5  12     12  13
119.7  91.3 133.2 127.6                             <- LINK ATTENUATIONS,
119.7 106.5  81.3 122.2                                   IN DB
 91.3 106.5  92.9 101.9  94.0
133.2  92.9 121.8 122.4  97.8
127.6 121.8 111.1 133.3
 81.3 101.9 103.7  97.6 113.2
 94.0 122.4 103.7 105.5  98.6  81.1
 97.8 111.1 105.5 110.9 130.0
122.2  97.6 123.4 117.6
113.2  98.6 123.4 131.2 118.6
 81.1 110.9 131.2 100.6 123.3
133.3 130.0 100.6 140.7
117.6 118.6 123.3 140.7
  1.0   2.0   3.0   5.0   6.0   7.0   9.0  11.0  13.0  15.0  <- ATTENUA-
 17.0  20.0  22.0  25.0  29.0  32.0  36.0  40.0  44.0  49.0  TION BIN
 55.0  60.0  66.0  73.0  81.0  88.0  97.0 107.0 117.0 128.0  ASSIGNMENTS
300.0      <- TEST.DURATION
1000       <- MAX.CKTS.IN SIM
12         <- SLOTS (PER FRAME)
1  4       <- MINIMUM SLOT STACKING DEPTH, MAXIMUM SLOT STACKING DEPTH
0.001      <- SLOT.DURATION
0.0001     <- PROCESSING.TIME
```

```
FILE: THESIS   DATA     A1  NAVAL POSTGRADUATE SCHOOL

0.00002   <- PROP.DELAY.TIME
6.5       <- MEAN TIME BETWEEN CIRCUIT REQUIREMENTS FOR EACH NODE
10.0      <- MEAN DURATION OF AN ESTABLISHED CIRCUIT
5.0       <- TIME PERIOD BETWEEN DYNAMIC ROUTING UPDATE CYCLES
15.0      <- TIME PERIOD BETWEEN PERIODIC STATUS REPORTS
0.0       <- PERCENTAGE OF CIRCUIT REQUIREMENTS GENERATED IN GROUP
0.0       <- PERCENTAGE OF CIRCUIT REQUIREMENTS GENERATED IN FAMILY
96        <- X COORDINATE OF THE BREAK.POINT
256       <- Y COORDINATE OF THE BREAK.POINT
1024.0    <- MAXIMUM SCALED NODE WEIGHT
 0  2  3  4  5  3  3  4  2  2  5  5  2  <- STATIC BEST.PATH ARRAY
 1  0  3  3  1  6  6  1  9  6  9  9  9
 1  2  0  4  1  6  7  4  2  6  7  4  6
 1  1  3  0  5  3  7  8  3  7  8  5  8
 1  1  4  4  0  1  4  8  1 12  8 12 12
 2  2  3  3  3  0  7  7  9 10 10 10  9
 4  3  3  4  8  6  0  8  6 10 11 11 10
 5  4  4  4  5  7  7  0 11 11 11 12 12
 2  2  6  2 13  6 10 13  0 10 10 13 13
 6  9  6  7 11  6  7 11  9  0 11 13 13
 8 10  7  8 12 10  7  8 13 10  0 12 13
 5  5  5  8  5 13  8  8 13 11 11  0 13
12  9  9 12 12 10 11 11  9 10 11 12  0
1239 4568 7897  126 3455 6784 9013 2342 5671 8900  <- SEVERAL LINES OF
1239 4568 7897  126 3455 6784 9013 2342 5671 8900       THE SAME RANDOM
1239 4568 7897  126 3455 6784 9013 2342 5671 8900          NUMBER SEED
1239 4568 7897  126 3455 6784 9013 2342 5671 8900             NUMBERS
1239 4568 7897  126 3455 6784 9013 2342 5671 8900
1239 4568 7897  126 3455 6784 9013 2342 5671 8900
1239 4568 7897  126 3455 6784 9013 2342 5671 8900
1239 4568 7897  126 3455 6784 9013 2342 5671 8900
1239 4568 7897  126 3455 6784 9013 2342 5671 8900
1239 4568 7897  126 3455 6784 9013 2342 5671 8900
1239 4568 7897  126 3455 6784 9013 2342 5671 8900
1239 4568 7897  126 3455 6784 9013 2342 5671 8900
1239 4568 7897  126 3455 6784 9013 2342 5671 8900
1239 4568 7897  126 3455 6784 9013 2342 5671 8900
1239 4568 7897  126 3455 6784 9013 2342 5671 8900
1239 4568 7897  126 3455 6784 9013 2342 5671 8900
1239 4568 7897  126 3455 6784 9013 2342 5671 8900
1239 4568 7897  126 3455 6784 9013 2342 5671 8900
1239 4568 7897  126 3455 6784 9013 2342 5671 8900
1239 4568 7897  126 3455 6784 9013 2342 5671 8900
/*
//
```

LIST OF REFERENCES

1. Lucke, E. A., An Investigation of Distributed Communications Systems and Their Potential Applications to the Command and Control Structure of the Marine Corps, Master's Thesis, Naval Postgraduate School, Monterey, California, December 1979.

2. Segall, A. and Merlin, P., "A Failsafe Distributed Routing Protocol", IEEE Transactions on Communications, v. com-27, no. 9, pp. 1280-1287, September 1979.

3. Bond, J. E., Self-Interference in a Packet Radio Network, Master's Thesis, Naval Postgraduate School, Monterey, California, December 1980.

4. Kane, T. G., Computer Investigation of VHF, UHF and SHF Frequencies for Marine Corps Packet Radio Usage, Master's Thesis, Naval Postgraduate School, Monterey, California, December 1980.

5. Hobbs, C. D., Selection of Packet Radio Links for Tactical, Distributed Communication Networks, Master's Thesis, Naval Postgraduate School, Monterey, California, September 1981.

6. Chlebik, M. G., A Study of Traffic Flow in a Marine Amphibious Brigade, Master's Thesis, Naval Postgraduate School, Monterey, California, December 1981.

7. Mercer, R. T., A Study of Interference in a Tactical Packet Radio Network, Master's Thesis, Naval Postgraduate School, Monterey, California, June 1982.

8. Lengerich, A. W., Investigations into the Performance of a Distributed Routing Protocol for Packet Switching Networks, Master's Thesis, Naval Postgraduate School, Monterey, California, December 1982.

9. Heritsch, R., A Distributed Routing Protocol for a Packet Radio Network, Master's Thesis, Naval Postgraduate School, Monterey, California, March 1982.

10. Dixon, R. C., Spread Spectrum Systems, Wiley, 1976.

11. Mowafi, O. A. and Kelly, W. J., "Integrated Voice/Data Packet Switching Techniques for Future Military Networks", IEEE Transaction on Communications, v. com-28, no. 9, pp. 1655-1662, September 1980.

12. Kuo, F. F., Protocols & Techniques For Data Communications Networks, Prentice-Hall, 1981.

13. Brady, P. T., "A Technique for Investigating On-Off Patterns of Speech", Bell System Technical Journal, v. 44, pp. 1-22, January 1965.

14. Bullington, K. and Fraser, J. M., "Engineering Aspects of TASI", Bell System Technical Journal, v. 38, pp. 353-364, March 1959.

15. Shanmugam, K. S., Digital and Analog Communications Systems, Wiley, 1979.

16. Haykin, S., Communications Systems, Wiley, 1978.

17. Yannoni, N. F., "Precision Frequency Standards Upgrade Performance of C^3I Systems", MSN, pp. 114-129, June 1982.

18. Unnamed author, "Crystal Oscillators Extend Frequencies", Military Electronics, p. 62, October 1982.

19. Marine Corps Tactical Systems Support Activity Technical Note 78-C-06, Multichannel Requirements Analysis, 1 December 1978.

20. Tannenbaum, A. S., Computer Networks, Prentice-Hall, 1981.

21. Gallager, R. G., "A Minimum Delay Routing Algorithm Using Distributed Computation", IEEE Transactions on Communications, v. com-25, no. 1, pp. 73-85, January 1977.

22. Dijkstra, E. W., "A Note on Two Problems in Connexion with Graphs", Numerische Mathematik, v. 1, pp. 269-271, 1959.

BIBLIOGRAPHY

Bondy, J. A. and Murty, U. S. R., Graph Theory With Applications, American Elsevier Publishing Company, 1976.

Carré, B., Graphs and Networks, Clarendon Press, 1979.

Consolidated Analysis Centers Inc. (CACI, Inc.), SIMSCRIPT II.5 Reference Handbook, 1976.

Kahn, R. E., Gronemeyer, S. A., Burchfiel, J., and Kunzelman, R. C., "Advances in Packet Radio Technology", Proceedings of the IEEE, v. 66, no. 11, pp. 1468-1496, November 1978.

Kim, B. G., Queueing Analysis of Scheduling and Flow Control Techniques in Integrated Voice and Data Packet-Switched Networks, Ph.D. Dissertation, University of Massachusetts, September 1980.

Kiviat, P. J., Villanueva, R., and Markowitz, H. M., SIMSCRIPT II.5 Programming Language, CACI, 1975.

Schwartz, M., Computer-Communication Network Design and Analysis, Prentice-Hall, 1977.

INITIAL DISTRIBUTION LIST

		No. Copies
1.	Defense Technical Information Center Cameron Station Alexandria, Virginia 22314	2
2.	Library, Code 0142 Naval Postgraduate School Monterey, California 93940	2
3.	Department Chairman, Code 62 Department of Electrical Engineering Naval Postgraduate School Monterey, California 93940	1
4.	Prof. J. M. Wozencraft, Code 62Wn Department of Electrical Engineering Naval Postgraduate School Monterey, California 93940	3
5.	Commanding General Marine Corps Development and Education Command C^3 Division, Development Center, D102D Attn: CAPT W. K. Tritchler Quantico, Virginia 22134	3
6.	Commanding General Marine Corps Development and Education Command C^3 Division, Development Center, D102D Attn: MAJ E. A. Lucke Quantico, Virginia 22134	1
7.	Commanding General Marine Corps Development and Education Command C^3 Division, Development Center, D102D Attn: CAPT T. G. Kane Quantico, Virginia 22134	1
8.	Captain Kenneth L. Moore, USAF, Code 62Mz Department of Electrical Engineering Naval Postgraduate School Monterey, California 93940	1
9.	Lieutenant Commander Anthony W. Lengerich, USN Staff CARGRUTWO FPO New York, New York 09501	1

10.	Captain R. R. Logan, USMC SMC 1231 Naval Postgraduate School Monterey, California 93940	1
11.	Lieutenant W. K. Tritchler, USN SMC 2961 Naval Postgraduate School Monterey, California 93940	1
12.	Marine Corps Representative, Code 0309 Naval Postgraduate School Monterey, California 93940	1
13.	Commander, Naval Ocean Systems Center Attn: Mr. D. Leonard, Code 8195 271 Catalina Boulevard San Diego, California 92152	1
14.	Dr. B. M. Leiner Information Processing Technology Office Defense Advanced Research Projects Agency 1400 Wilson Boulevard Arlington, Virginia 22209	1
15.	Colonel H. L. Fogarty, USMC (Ret.) 39331 Chaparral Drive Glenoak Hills Temecula, California 92005	1

200780

Thesis
T8123 Tritchler
c.1 A time slot assignment
algorithm for a TDMA
packet radio network.